An Infinite Key Encryption System

SOFTWARE FOR ADVANCED PROGRAMMERS

Dr. Dobb's Journal

SCISTAR **Greek and Math Symbols** with WORDSTAR

File Comparator for CP/M Plus

A Small-C **Concordance** Generator



"dBASE II is far, far better than a shoehorn."

Rusty Fraser President Data Base Research Corp.

"We laughed when our customers asked us to put our minicomputerbased real-time accounting system, The Champion," on a micro.

"No way was it going to fit, we thought.

"We'd have to create our own database management system and, even then, it'd be a tight squeeze.

"Then we discovered dBASE II, the relational database management system for microcomputers from Ashton-Tate."

"dBASE II was a perfect fit."

"dBASE II is a program developer's dream come true. The dBASE II RunTime™ module quickly provided us with the powerful text editing, data entry speed and other 'building block' capabilities we needed to develop and deliver a new Champion to our customers—the leading real-time on-line accounting system available for a micro."

The short cut to success.

The dBASE II RunTime module has helped a lot of program devel-



opers like Data Base Research become successful software publishers.

For more about dBASE II and RunTime, contact Ashton-Tate 10150 West Jefferson Boulevard, Culver City, CA 90230, (800) 437-4329, ext. 217. In the U.K., call (0908) 568866.

For more about The Champion, call Data Base Research at (303) 987-2588.



dBASE II is a registered trademark and RunTime is a trademark of Ashton-Tate. The Champion is a trademark of Date Base Research Corporation.
© Ashton-Tate 1983.

Professional BASIC[™] cuts your program debugging time in half. Or your money back.

ntroducing Professional BASIC™. A powerful new BASIC programming language that can help you debug, or learn—and then master—the language. Professional BASIC can actually double your programming efficiency. Or your money back.*

What You See Is What You Get.

Because Professional BASIC has a sophisticated window-oriented system environment—containing more than a *dozen* visualization and debugging screens—you see *clearly and precisely* the execution of your program. Line by line pro-



gram execution unfolds via a dynamic program trace...showing changes in variables or array elements... or the progress of FOR/NEXT loops or GOSUBS. Using Professional BASIC, even beginning programmers will be able to produce code quicker, with far less frustration.

Now, Access All The Memory Of Your PC.

Professional BASIC can use all of the mem-





Personal Computer. (If expansion boards are used, that means up to 640K.) Now you can create and run programs almost as big as your imagination—and talent.

HIHHHHHH

The 8087 Connection. Accurate, High - Speed Math.



signed to use the ultrafast 8087 math coprocessor.
The 8087 allows floating-point math operations to be performed with lightning speed. Professional BASIC enables you to fully utilize the 8087—to dramatically accelerate execution of programs involving complex floating-point computations. (8087 coprocessor optional.)

Read What The Experts Say.

"This version of BASIC sets new



standards for usability and 'user friendliness'." "Morgan's product supports, in an unprecedented way, the visualization of program execution."

> April 1984 Byte Magazine

"The real magic of Professional BASIC is its wealth of 'windows' into an executing program." "I'm frankly amazed. My hat is off to Dr. Bennett... An elegant piece of coding indeed."

February 1984 Personal Computer Age

"This version of the language sets new standards..." "The user interface... is unbeatable for program development." "... You owe it to yourself to pick up the phone and order a copy of Professional Basic."

June 1984 Dr. Dobb's Journal

Makes Learning The BASICs A Snap.

Learn more about the BASIC language... in significantly less time. With Professional BASIC there are over 18 different ways to view your program as it operates, step by step. On a split screen display you see, on one side, a dynamic display of your executing program's code...on the other side vou actually see changes in variables as they occur.

PROFESSIONAL BASIC SPECIFICATIONS

IBM PC or XTTM, Compaq[™], or 100% compatibles.

DOS 2.x operating system.

1 Double-sided disk drive.

256K RAM minimum (384K recommended)

8087 Chip optional

Ask your local dealer for a demonstration, or call or write Morgan Computing Co., Inc. at 10400 N. Central Expressway, Suite 210, Dallas, Texas 75231 (214/739-5895). Suggested retail is \$345.00. (Dealer inquiries welcome.) Demo disk is \$5.00 (includes demo on TRACE86™ and TED™).

*Sole remedy for failure of product to double your debugging speed is return of full purchase price. To obtain a refund you must act within thirty (30) days of purchase (i.e., return both product and original sales receipt) and have completed and filed all warranty registrations with MCC. Further, you must reaffirm you have not copied the product or violated any copyright of MCC. Contact MCC for additional information. Offer expires November 1, 1984.



Morgan Computing Co., Inc.

Software Designed By Professionals. For Professionals.

COHERENT™ IS SUPERIOR TO UNIX* AND IT'S AVAILABLE TODAY ON THE IBM PC.

Mark Williams Company hasn't just taken a mini-computer operating system, like UNIX, and ported it to the PC. We wrote COHERENT ourselves. We were able to bring UNIX capability to the PC with the PC in mind, making it the most efficient personal computer work station available at an unbelievable price.

For the first time you get a multi-user, multitasking operating system on your IBM PC. Because COHERENT is UNIX-compatible, UNIX software will run on the PC under COHERENT.

The software system includes a C-compiler and over 100 utilities, all for \$500. Similar environments cost thousands more.

COHERENT on the IBM PC requires a hard disk and 256K memory. It's available on the IBM XT, and Tecmar, Davong and Corvus hard disks.

Available now. For additional information, call or write,

Mark Williams Company 1430 West Wrightwood, Chicago, Illinois 60614 312/472-6659



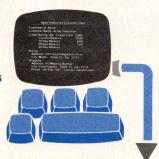
COHERENT is a trade mark of Mark Williams Company. *UNIX is a trade mark of Bell Laboratories.

ALL AT ONCE!

AND NEVER A "LOCKED OUT" USER!

1. Accounts Receivable Manager performs a customer query and has DataFlex and has DataFlex print a report for each account with a balance over 60 days.

2. Billing clerk makes change of billing address.





3. Sales Secretary receives change of receives change notice phone number notice accesses record to accesses the phone update field.



Apex Industries Customer Data

Customer Id: Acme

Customer Name: Acme Industries

Credit Rating: AA Credit Limit: 25000 Current Balance: 12500

30 Day Balance: 4000 60 Day Balance: 1500 90 Day Balance: 0

Billing:

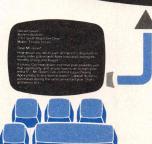
Address: 2701 South Bayshore Drive City: Miami State: FL Zip: 33133

Shipping:

Address: 913 Majorca Avenue

City: Coral Gables State: FL Zip: 33134
Phone: 305-856-7503 Contact: Gerald Green

6. Advertising promotion Director orders tion Director orders a mailing to the cona mailing to the custact at each custact at each customer, merging tomer, merging tomact name and contact name and contact name and contact name and contact of the custact at each cuscusword processor preword processor preword document.







5. Apex's salesman on the Acme account the Acme account the Acme as a sale and makes a sansaction posts a transaction which updates the which updates the Current Balance field Current Balance field Acme's record.

DataFlex is the only application development database which *automatically* gives you true multi-user capabilities. Other systems can lock you out of records or entire files for the full time they are being used by someone else. DataFlex, however, locks only the data being changed, and *only* during the micro-seconds it

takes to actually write it to the file! The updated record is then immediately available. The number of users who can access, and change, records at the same time is limited only by the number of terminals on your system or network. Call or write today for all the details on DataFlex...the true multi-user database.

DATAFLEX

DATA ACCESS CORPORATION

8525 SW 129 Terrace, Miami, FL 33156 (305) 238-0012 Telex 469021 DATA ACCESS CI

Compatible with CP/M-80, MSDOS networks, MP/M-86, Novell Sharenet, PC-Net, DMS Hi-net, TurboDOS multi-user, Molecular N-Star, Televideo MmmOST, Action DPC/OS, IBM PC w/Corvus, OMNINET, 3Com EtherSeries and Micromation M/NET.

MSDOS is a trademark of Microsoft. CP/M and MP/M are trademarks of Digital Research.

Dr. Dobb's Journal

Editorial

Editor Managing Editor **Contributing Editors**

Editor-in-Chief Michael Swaine Revnold Wiggins Randy Sutherland Robert Blum,

Dave Cortesi, Ray Duncan, Anthony Skjellum, Michael Wiesenberg

Copy Editors Polly Koch, Cindy Martin

Production

Typesetter Jean Aring

Design Director Fred Fehlau

Art Director Shelley Rae Doeden Production Manager Detta Penna

Production Assistant Advertising

Alida Hinton Cover Artist Claudia Majewski

Advertising Sales Alice Hinton,

Walter Andrzejewski

Circulation

Circulation and Promotions Director Fulfillment Manager **Direct Response**

Beatrice Blatteis Stephanie Barber

Coordinator **Promotions Coordinator** Single Copy Sales

Maureen Snee Jane Sharninghouse

Coordinator Single Copy Sales Circulation Assistant Kathleen Boyd

Lorraine McLaughlin

M&T Publishing, Inc.

Publisher and Chairman of the Board Otmar Weber Director

C.F. von Ouadt President Laird Foshay

Entire contents copyright @ 1984 by M&T Publishing, Inc. unless otherwise noted on specific articles. All rights reserved.

Dr. Dobb's Journal (USPS 307690) is published monthly by M&T Publishing, Inc., 2464 Embarcadero Way, Palo Alto, CA 94303, (415) 424-0600. Second class postage paid at Palo Alto and at additional entry points.

Address correction requested. Postmaster: Send Form 3579 to Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303. ISSN 0278-6508

Subscription Rates: \$25 per year within the United States, \$44 for first class to Canada and Mexico, \$62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Contributing Subscribers: Christine Bell, W.D. Rausch, DeWitt S. Brown, Burks A. Smith, Robert C Luckey, Transdata Corp., Mark Ketter, Friden Mailing Equipment, Frank Lawyer, Rodney Black, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor Jr., Ted A. Reuss III, InfoWorld, Stan Veit, Western Material Control, S.P. Kennedy, John Hatch, Richard Jorgensen, John Boak, Bill Spees, R.B. Sutton. Lifetime Subscribers: Michael S. Zick, F. Kirk.

Foreign Distributors: ASCII Publishing, Inc. (Japan), Computer Services (Australia), Computer Store (New Zealand), Computercollectief (Nederland), Homecomputer Vertriebs GMBH (West Germany), International Presse (West Germany), La Nacelle Bookstore (France), McGill's News Agency PTY LTD (Australia), Progresco (France).

People's Computer Company

Dr. Dobb's Journal is published by M&T Publishing, Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a non-profit, educational corporation.

August 1984 Volume 9, Issue 8

CONTENTS

In this Issue

This month's articles focus primarily on text files, from protection to comparison to indexing to printing. Our cover article presents a public-domain, infinitekey encryption system for CP/M, with discussion ranging from the theory behind constructing a secure encryption system to practical considerations that can affect the design. For those interested in trying out (or trying to break) this system, the code and documentation should be available from a number of sources by the time you read this. In addition to obtaining a disk from the authors, a couple of places you should find it are on Byron McKay's PicoNet RCPM (415-965-4097) and Mel Cruts' RCPM (408-263-2588). We'll update you periodically on where else it may be obtained through the Letters to the Editor column.

Upcoming Special Issues

Next month we present our fourth annual Forth issue. As in past years, the response has been good, and you should find it useful and educational. Unix authors should note that we are planning a special Unix issue for this December. Those interested may contact us by phone or mail to discuss topics. We should receive manuscripts by September 7, 1984. We prefer receiving manuscripts via magnetic or electronic media, but we are still building our repertoire of capabilities. Contact us to coordinate disk formats or electronic services.

A Brief Reminder

We have already begun receiving entries for the Doctor's Fifth Generation Programming Competition. Those who wish to participate should remember that entries should be postmarked no later than September 31, 1984. If this is the first you have heard of it and would like details, give us a call or look in the May and June issues of DDJ.

This Month's Referees

Dr. Dobb's Journal regularly draws on the expertise of a Board of Referees for technical evaluation of material submitted for publication. In addition to remarks to the editors concerning accuracy and relevance on manuscripts, the referees often provide constructive comments for authors regarding clarity or completeness. Their remarks help prevent authors from exposing blindspots or misconceptions in print and help ensure that our readers receive clear and accurate information. The referees who contributed to this month's issue are:

Wayne Chin, Hewlett-Packard John P. Keyes, Syracuse University

John K. Taber, IBM

Dr. Dobb's Journal

ARTICLES

- SCISTAR—Greek and Math Symbols 26 Describes how to implement a universal set of userwith WordStar programmed printer commands using WordStar with by John Thomas and Guy Fletcher MailMerge (Reader Ballot No. 191) What's The Diff?—A File Comparator How to design and implement a file difference finder in for CP/M Plus Pascal that saves the differences between two versions as a by D.E. Cortesi script of ED commands (Reader Ballot No. 192) **Designing a File Encryption System** How to build a strong, practical infinite-key encryption by John A. Thomas and Joan Thersites system for CP/M, including discussion on background,
 - A Small-C Concordance Generator by John E. Staneff
- 86 Extending an internal sort capability by using word lists during the sort input phase, this program extracts desired words from an input file and produces an index to their source (Reader Ballot No. 194)

theory, and design tradeoffs (Reader Ballot No. 193)

DEPARTMENTS

Editorial 6

Letters 8

Dr. Dobb's Clinic 22 Transcendental Sources; Over-Submitting; An Old Bug; BASIC Precision, Again (Reader Ballot No. 190)

16-Bit Software Toolbox

106 More on Savage's benchmark: An updated table of results, some timings on spreadsheets and Modula-2, and a discussion on possible benchmark weakpoints (Reader Ballot No. 195)

C/Unix Programmer's Notebook

116

uucp, public domain C, a standard fopen and fclose for BDSC, and some comments about C I/O and Unix comments (Reader Ballot No.196)

Software Reviews 122 PC-Write, PC Small-C 2.0

Of Interest 126 (Reader Ballot No. 197)

Advertiser Index 128



he Apple Macintosh is frustrating. Its major use right now seems to be for drawing pictures. Macwrite is slow. The amount of disk swapping the machine requires is unacceptable. The machine forces users to learn to operate a new control, the mouse. (MicroPro president Seymour Rubinstein says of the mouse that it's great for people with three hands.) And the Mac has no color

If you write software or build hardware, the Mac is a source of particular frustration. To design software for it you have to buy a Lisa, and to buy a Lisa you have to sit on a rubber waiting list that always seems to be three months long. Hardware manufacturers can't do much with that closed box: you need a footlong hex driver just to open the case, there are no slots and it takes courage to imagine modifying a four-layer motherboard. (Nevertheless, some nerveless hardware hackers are already creating their own 512K fat Macs.)

Despite all these frustrations, it remains the case that the Apple Macintosh has changed the entire nature of the game. It has redefined the computer-person interface, so radically altering the way people deal with computers that we can't go back to the simpler days. It makes the IBM PC look like a Processor Technology Sol.

Go into a computer store that has a Mac in stock and you'll see novices approaching and using the machine without fear or prodding. You'll hear people who claim not to like computers grudgingly admitting they like this one.

The Mac was designed with more thought for the user than any other personal computer. Consider the windows: there are window implementations for the IBM PC and other machines; why are they unsatisfying? Author Cary Lu thinks it's because they are still dependent on character-mapped graphics. The result: "the 'desktop' looks cluttered," according to Lu. The Mac's display design principle (everything is graphics) makes it simple to implement various character fonts, and the different fonts make the different window displays distinctive. So Mac windows are beautiful and natural, while windows on the PC look cluttered and kludgy.

Yes, the Mac lacks cursor keys, and the Model T lacked reins. Keyboards are good for typing letters and numbers. To imagine that they are the best way to represent movement in 2-space is to exhibit remarkable naivity.

Yes, MacWrite is slow. Some programmers recognize that inadequacy as an opportunity, and one of them will probably get rich from the observation. Better word processors will supplant the first version of MacWrite. At least two are in the works at Apple, but the eventual winner is likely to come from outside. Apple won't mind.

Apple (and, yes, Xerox PARC where the ideas originated) put considerably more thought into the Mac/Lisa user interface than MITS put into the Altair bus in 1974. But just as MITS's competitors took the Altair bus away from MITS and refined it into the S-100 bus, good programmers can make the Mac/Lisa user interface their own. Can, and will.

Probably just about the time some new development makes it obsolete.

Michael Swaine

Michael Swans

LIFEBOAT Associates: Software with Full Support programming horizon Reach for the of the 80's with Lattice® C. the fastest C compiler. Lattice C is available for a wide variety of 16-bit personal computers including IBM 16-bit personal computers including law.

NCR® Texas Instruments.

NCR® Texas Instruments. other microcomputers running PC MDOS, MS MARCH Call LIFEBOAT of 800-847-7078, or in N.Y. 212.860-0300, for free information on the Set the course of your next software project DOS and CP/M86. C.CHEST family of software development towards greater power and portability by rowards greater power and portuguity by selecting Lattice C, recognized as the finest selection Lattice C, recogni selecting Lattice C, recognized as the timest and fastest 16-bit C compiler. Lattice C, the full and rustest local of Kernighan and Ritchle, implementation of Kernighan and Ritchle, implementation or nernigram and kirchie full continues to lead the way by announcing full IFEBOAT memory management up to 1 Megabyte. memory management up to I megapyre.

Major software houses including management up to I megapyre. tools. Micropro and Sorcim are using Lattice C to Associates Lifeboat offers Lattice C with a tested set develop their programs. of software tools to provide a complete Color graphic primitives Customizable program editor development system including: Screen design aid Overlay linkage editor Screen and I/O utilities

LIFEBOAT" **Associates**

1651 Third Avenue New York, NY 10128

800-847-7078 212-860-0300

Please send me free information on:

C-FOOD SMORGASBORDM

LATTICE WINDOWM

- ☐ Lattice and development tools
- ☐ How to get your software published ☐ Corporate purchase program
- ☐ Dealer program

HALOM PANELTM

PMATEM PLINK M. 86

FOAT87

- ☐ OEM agreements
- ☐ Send me the complete LIFEBOAT software catalog. \$3.00 enclosed for postage and handling.

LIFEBOAT, Software with Full Support. C-CHESTIM Lifeboot Associates LATTICE C-FOOD SMORGASBORD and LATTICE WINDOW,TM Lattice, Inc.

Company

Address

State

Zip

Telephone

HALO, TM Media Cybernetics PANEL TM Roundhill Computer, Ltd. PMATE and PLINK,TM Phonix Software FLOAT87,TM Microfloat

Multi-window functions 8087 floating point math

> IBM and PC, ®TM International Business Machines MS.TM Microsoft CP/M86,TM Digital Research

Circle no. 33 on reader service card.

A String Finding Function

Dear Editor:

I very much enjoyed the article "Optimizing Strings in C" by Edward McDermott in the April '84 issue (*DDJ* #90).

One of the several additional routines suggested was one for locating strings. I needed such a routine to run under the Software Toolwork's uptown sister of Small C, C/80. Since it passes its arguments on the stack in the same fashion that Mr. McDermott's listing appears to expect them, I thought that the accompanying Listing One (page 10) might be of value.

At the time it was developed, I needed a routine for locating a string in a CP/M file placed in RAM, thus the string to be searched is called a "Character file" and the default terminator is CP/M's end of file 1Ah. If EOF is defined otherwise in the program, any terminator such as 0 may be used.

Sincerely, Paul C. Barton Col. Wilkins Rd. RD 1 Milford, NH 03055

Improved Directory

Dear DDJ:

I have enjoyed the Dan Daetwyler article "Sorted Diskette Directory for the IBM PC" (January DDJ, #87). One of the advantages of implementing something of that sort is the opportunity to add a few personal flourishes. I enclose two additions that do additional tasks, and a third which corrects an annoyance.

First, although one may not want hidden files included on the COVER, I do want to know that there are such files on the diskette, and how many. I have added code to count them and indicate the count at the end of the list.

Second, I find it very convenient to group like files together—e.g., list all

.COM files first (alphabetically), then all .BAS files, and lastly all the other files. I have included a procedure (COORDR) that prompts for the file name extensions that you wish to place first on the sorted list, and changes in COSCAN and COPRNT that handle the modified sort and print.

Finally, I note that if you have a file with the maximum 8-character file name plus maximum 3-character extension, and a file length greater than 99999, there is no space in the listing between name and length. I have added a single space between each column to take care of this.

I enclose complete listings (Listing Two, page 11) of the added procedure COORDR and the two procedures which have significant additions (COS-CAN and COPRNT, Listings Three and Four, pages 12 and 16). I also include full listing for COENDP (Listing Five, page 21) since, although there are only minor changes, it is short. In addition to the listings included, you need to add "GETORD:NEAR" to the EXTERN definition in the main program COVER, and the code line "CALL GETORD" following the "CALL GETTTL" (line 79 in the published listing of COVER). Finally, add code to make the PROC DECMAL in COTITL a PUBLIC definition.

Yours truly,
Bruce F. Cameron
4067 Rose Hill
Cincinnati, OH 45229

Relief for User Frustration

Dear Editor:

As you are no doubt aware, a problem of great dimension has surfaced in the computer industry, which is a cause of major concern for growing thousands of computer users. I speak of the problem of user frustration.

An individual who purchases a computer from a local dealer sometimes gets sold a "bill of goods." Even if the

dealer has been completely honest and aboveboard, the average user faces the problem of climbing a veritable Everest of manuals and other documents in an attempt to put the new purchase to work. I think you will agree with me that a great percentage of such persons experience teeth-grating frustration.

What to do? I am proposing a practical solution to the problem. An article that appeared in the April 2 issue of InfoWorld described a "movement" I am attempting to organize. Calling ourselves the Society for the Prevention of Cruelty to Prospective Computer Purchasers (SPC->PCP), our purpose is to link people with problems to people with solutions.

I have found through 20 years of experience in the computer industry that technical people are very friendly, on the whole, and willing to help others. I am trying to locate a large number of such technical people who would be happy to share their expertise with neophytes. Once I have accumulated such a list of experienced helpers, I shall advertise in computer magazines that such a list exists, and for a small fee (to cover the cost of the ads), anyone may send for a list of members in his or her immediate area.

The response I have been getting is overwhelmingly enthusiastic. The technical community is more than willing to support such an endeavor. In the process, of course, they may make relationships with some very important contacts. Everyone benefits!

I am hoping that, as a concerned member of the community of computer professionals, your magazine will lend support to SPC->PCP by helping me to contact the experienced technical individuals needed to offset the wave of frustration that is growing larger with each passing day. Your assistance is urgently needed.

Thank you very much.

Yours very truly, Burton Bhavisyat SPC->PCP
Rte 1, Box 318
Moundsville, WV 26041

DRI Indifferent?

Dear Editor:

I have never written a letter to *DDJ* before although I am a charter subscriber and I still have every issue since number 1. I have always admired your courage in publishing the facts to protect the public.

I feel it is necessary to warn users of an attitude at Digital Research which I will (with restraint) call "unabashedly indifferent." We used to accuse a particularly large mainframe manufacturer of this attitude, but it appears that Digital Research has fallen to new depths in the same attitude. Let me recount one small example, which will hopefully help others too.

We bought the Manx C compiler version 1.05G for \$200. It had several bugs. We were promised version 1.06 would solve those bugs—unfortunately it cost us \$250 to upgrade to version 1.06 two months after buying 1.05. That's bad enough. Version 1.06 also had bugs: some error messages had misspelled words in them and other very minor bugs.

The major problem was that the Manx linker crashed when trying to link a program which was correctly linked by 1.05. Version 1.06 advertised the ability to use the Digital Research RMAC/LINK programs. We said, great, we will use those. Apparently, no one at Manx had actually tried that—the compiler generated underscores and periods in names which the RMAC compiler could not handle. Manx has not yet communicated a fix for their linker, but did get me a patch after frequent calls (about six) and approximately two weeks. This did hold us up significantly in our development, but at least Manx did respond.

The patch was not for their compiler; instead, it was for RMAC. The patch, only a few lines, made RMAC accept underscores and periods. We discovered several bugs at that point with Digital Research's LINK—the worst was sprinkling of addresses on top of code when both ASEGs and CSEGs were used in the same area of memory. After we figured that out (it

Squeezing A Large Program Into A Small Memory Space?



It's time you got Plink86,™ the overlay linkage editor that's bringing modular programming to Intel 8086-based micros.

With Plink86,* you can write a program as large and complex as you want and not worry about whether it will fit within available memory constraints. You can divide your program into any number of tree-structured overlay areas. 4095 by 32 deep. Work on modules individually. Then link them into executable files. All without making changes to your source program modules.

Use the same module in different programs. Experiment with changes to the overlay structure of an existing program. Use one overlay to access code and data in other overlays.

Plink86 is a two-pass linkage editor that links modules created by the Microsoft assembler or any of Microsoft's 8086 compilers. Plink86 also works with other popular languages, like Lattice C, C86, or mbp/COBOL. And supports symbolic debugging with Phoenix' Pfix86 Plus.™

Plink86 includes its own object module library manager – Plib86™ – that lets you build libraries from scratch. Add or delete modules from existing libraries... Merge libraries... Or produce cross-reference listings.

Why squeeze any more than you have to? Plink86 by Phoenix. \$395. Call (800) 344-7200, or write

Phoenix

Phoenix Computer Products Corporation

1416 Providence Highway, Suite 220 Norwood, MA 02062 In Massachusetts (617) 769-7020

*Plink86 will run under PC DOS, MS-DOS™ or CP/M™-86.

Plink86, Pfix86 Plus and Plib86 are trademarks of Phoenix Software Associates Ltd. MS-DOS is a trademark of Microsoft Corporation. CP/M is a trademark of Digital Research, Inc.

wasn't easy), we avoided the constructs which caused them. The problems could not be demonstrated with a small test program (apparently due to table overflows?), therefore we didn't bother DR, knowing the difficulties in dealing with them.

We finally came across a bug in RMAC (very difficult to find) which we traced back to the original non-patched RMAC (version 1.1). The following program demonstrates it:

REPT 110 DB 0 ENDM END

The DB statement gets repeated 78 times! After some experimentation we discovered that all numbers between 97 and 122 were reduced by 32. This, of course, in retrospect, is an unfortunate attempt by RMAC to convert lower case 'a' to 'z' to upper case. Since this statement is used frequently by the

Manx C compiler, and the problem was so easily demonstrated, we decided to call Digital Research.

We subscribe to Digital Research's \$250 software maintenance program because we learned long ago that their regular maintenance line is perpetually busy (I have an automatic dialer). To make a long story slightly shorter, after four days of phone calls, all the way to the head of marketing, their attitude seemed to be:

- 1. That DR doesn't support 8-bit products anymore, because they don't make DR money, and
- 2. That even if they decided to fix it, it would be a year or so.

What capped it off was that the head of marketing told us we might want to purchase the \$250 maintenance package. That was the straw that broke the camel's back. When we informed him we currently did subscribe, he said, "Oh, maybe we can fix it in a year, but don't count on it."

In summary, Manx is a relatively

small company. Their attitude is understandable, they aren't millionaires, and they do try. Digital Research seems to have gotten fat and sassy and said to heck with the user and his problems. A bug is a bug and should be fixed.

This is only one of dozens of problems I have had with DR since I started using CPM V1.3 in 1976. I have implemented the BIOS/XIOS for V1.3, V1.4, V2.2, V3.0 and MPM V1.1, V2.1, found many bugs, never had any of them fixed or replied to, written many letters to DR, and received *no* replies.

This was such a simple bug that caused so much trouble, I just had to write to warn potential DR users what I think they can expect.

Sincerely,
Steve Conley
2030 Powers Ferry Rd.
Suite 110
Atlanta, GA 30339

*/

Letters Listing One

/* The function char *findstr(is, cf) returns a pointer to the first occurence of string pointed to by is in that pointed to by cf.

In the default case, IS is terminated by a 0 byte and CF is terminated by CPM's end of file 26 (^Z).

To change these defaults, the calling program must define the constants EOS (end of string for is) and EOF (end of file for cf).

```
#ifndef EOS
#define EOS 0
#endif
#ifndef EOF
#define EOF 26
#endif
/*******
char *findstr(is, cf)
char *is, *cf;
-{
#asm
                 POP B
                                 ;Return address
                 POP D
                                 ; Character file -cf
                                 ; Index string - is
                 POP H
;
                                 ; Restore the stack for caller
                  PUSH H
                  PUSH D
                  PUSH B
```

```
SEARCH:
                        LDAX
                                D
                                            ; *CF to A
SEARCH1:
                        CMP
                                M
                                            ; *CF == IS[0] ?
                        CZ
                                INDEX
                                            ; If yes call (check rest of IS)
                        INX
                                D
                                            ; No match ++CF
                        LDAX
                                D
                                            ; Check for end of CF
                        CPI
                                EOF
                        JNZ
                                SEARCH1
                                            ;Loop if not
FAIL:
                        LXI
                                D,0000
                                            ; Search failed.
                                                                  Load error message.
                        JMP
                                DONE
                                            ; And go back to caller
INDEX:
                        PUSH
                                D
                                            ; Save registers
                        PUSH
                                H
INDEX1:
                        INX
                                D
                                                                    We know the last one matched
                                            ; Next character.
                        INX
                                H
                       MVI
                                A, EOS
                                            ;Check for end of IS
                       CMP
                       JZ
                                            ; If found match is complete.
                                SUCCESS
                       LDAX
                                D
                                            ; Not yet
                       CMP
                                M
                       JZ
                                INDEX1
                                            ; This character matched. Loop for next.
                       POP
                                            ; No match.
                                                           Restore registers
                       POP
                               D
                                            ; in preparation for
                       RET
                                            ; return to search of CF for match of IS[0]
SUCCESS:
                       POP
                               H
                       POP
                               D
                                           ; Restore pointer to first position of IS in CF
                       POP
                               B
                                           ; Pop call from search. Now will ret to caller
DONE:
                       XCHG
                                            ; Pointer into HL for return.
#endasm
                                                                                            End Listing One
Listing Two
                                  PAGE
                                  TITLE
                                         COORDR
                                                   Diskette contents list - Redefine Order
                                  COMMENT
                                                       * Added PROC - Feb 1984 *
                                  PAGE
                                  Request file extension entries which are to be sorted to the beginning of the list
0000
                           CODE
                                  SEGMENT PARA PUBLIC 'CODE'
                                  ASSUME CS:CODE, DS:CODE
                                  EXTRN
                                         RELOC: BYTE
0000 80
                           SPCHAR
                                  DB
                                         80H
                                                              ; Ordering character
0001
     08 00
                                         8,0
8 DUP (?)
                           XBUF
                                  DB
                                                              ; Input buffer for extension
0003
       1 80
                                  DB
                1
000B
     OD OA 45 6E 74 65
                           XPRMT1 DR
                                         13,10, 'Enter file name extension to relocates'
     72 20 66 69 60 65
     72 26 61 6B 65 20
65 78 74 65 6E 73
69 6F 6E 20 74 6F
20 72 65 6C 6F 63
61 74 65 24
     0D 0A 20 20 20 74
6F 20 74 6F 70 20
0033
                           XPRMT2 DB
                                         13,10,
                                                 to top of list ("\" to quit): $'
       66 20 60
20 28 22
               69
50
                  73
22
     6F
     74
```

(Continued on next page)

Letters (Listing continued, text begins on page 8) **Listing Two**

```
PUBLIC
                                              GETORD
                              GETORD
                                              NEAR
0057
                                      PROC
                                                                     ; Counter of entries
0057
     RD 0000
                                      MOV
                                              BP,0
                                              DI, OFFSET RELOC
005A
     BF 0000 E
                                      MOV
005D BA 000B R
                              LOOP:
                                      MOV
                                              DX, OFFSET XPRMT1
                                                                     ; Prompt for extension
                                              AH, 9
21Ĥ
                                      MOV
0060 B4 09
0062 CD 21
                                      INT
                                              DX, OFFSET XPRMT2
0064
     BA 0033 R
                                      MOV
                                              AH, 9
21H
     84 09
0067
                                      MOV
0069
     CD 21
                                      INT
                                              DX, OFFSET XBUF
006B BA 0001 R
                                      MOV
                                              AH, OAH
                                                                      ; Get extension
006E B4 0A
                                      MOV
0070
     CD 21
                                              21H
                                      INT
0072 B0 3E 0003 R 5C
0077 74 32
0079 BA 1E 0002 R
0070 80 FB 02
                                              XBUF+2,5CH
                                                                      ; Check for done (\)
                                      CMP
                                      JE
                                              TIUG
                                              BL, XBUF+1
BL, 2
                                      MOV
                                                                      ; Get length
                                      CMP
                                              STOR
0080
     7F 13
                                      JNLE
0082
      32 FF
                                       XOR
                                              BH, BH
                                              CX, BX
0084
      8B CB
                                       MOV
                                              CX,3
      83 E9 03
                                      SUR
0086
                                                                      ; Set loop count (3 - length of string)
                                              CX
0089 F7 D9
                                      NEG
008B
008C
                                      PUSH
                                              DI
      57
                                              DI, XBUF[BX+2]
                                                                      ; Points to char just after string
                                      LEA
      8D BF 0003 R
                                                                      ; Fill with blanks
0090
      BO 20
                                       MOV
                                                                          to 3 characters
                                       REP
                                              STÓSB
0092
      F3/ AA
                                       POP
                                              DI
0094
      5F
                                              CX.3
SI,OFFSET XBUF+2
                               STOR:
                                       MOV
0095
      R9 0003
0098
      BE 0003 R
                                       MOV
                                                                      ; Extension from buffer to list
                                       REP
                                              MOVSB
009B F3/ A4
009D A0 0000 R
                                       MOV
                                              AL, SPCHAR
                                                                      ; Add 'ordering' character
                                       STOSE
AA 0A00
                                               SPCHAR
                                       INC
00A1 FE 06 0000 R
                                                                      ; Count entry
                                               BP
00A5
                                       INC
     45
     83 FD 0A
                                       CMP
                                               BP, 10
00A6
                                                                      ; Cycle if 9 or fewer entries
      7C B2
C6 05 00
                                               LOÓP
                                       JL
00A9
                                       MOV
                                               BYTE PTR [DI],0
                                                                      ; Terminator to end-of-list
                               QUIT:
OOAB
                                                                      ; Reset ordering character
      C6 06 0000 R 80
C3
OOAE
                                       MOV
                                               SPCHAR, 80H
                                       RET
00B3
                               GETORD
                                       ENDP
00B4
                               CODE
                                       ENDS
0084
                                       END
Segments and groups:
                                                       combine class
                Name
                                        Size
                                                align
                                        0084
                                                PARA
                                                        PUBLIC 'CODE'
Symbols:
                Name
                                        Туре
                                                Value
                                                        Attr
                                                        CODE
 GETORD . . . . . . . . . . . .
                                        N PROC
                                                0057
                                                                Global Length =005D
                                                0050
                                        L NEAR
                                                        CODE
L NEAR
                                                BAGO
                                                        CODE
                                        V BYTE
                                                0000
                                                        CODE
                                                                External
                                        L BYTE
                                                0000
                                                        CODE
 L NEAR
                                                0095
                                                        CODE
 XBUF
                                        L BYTE
                                                0001
                                                        CODE
 CODE
                                        L BYTE
                                                0000
                                        L BYTE
                                                        CODE
                                                0033
 Warning Severe
 Errors Errors
```

End Listing Two

(Listing Three begins on page 14)

——PRESENTING—— The first compiler for dBASE II*



WordTech Systems is proud to announce the first compiler for dBASE II[®]. And we are introducing it with a special offer.

-INDEPENDENCE

Now you can write compiled, efficient programs that will execute independently of dBASE II, and without RunTime®.

NO LICENSE FEES-

You only buy dB Compiler™ once. You may compile as many applications as you wish, FOREVER, with no additional fees.

SPEED

Application programs are compiled into low level code and only include program functions that are absolutely necessary.

SECURITY-

Compilation is far better than encryption for protecting your programming insights and procedures.

-PORTABILITY-

Using dB Compiler's cross-linkers you can use one development system to generate code for various target environments.

Suggested retail price: \$750; additional target modules: \$350 Special Offer: Compiler and an additional target module: \$750 Offer expires 7/15/84. Corp/multi-user licenses available.



WORDTECH SYSTEMS P.O. Box 1747, Orinda, CA 94563 (415) 254-0900

dBASE II. RunTime® Ashton-Tate

Letters (Listing continued, text begins on page 8) **Listing Three**

```
.132
COSCAN
                                           PAGE
                                                                Diskette contents list - Scan Directory
                                           TITLE
                                                            * Version 1.0 - June 1983
                                           COMMENT
                                                               Modified Jan 1984 *
                                                    82
                                           PAGE
                                           SEGMENT PARA PUBLIC 'CODE'
                                  CODE
0000
                                           ASSUME
                                                   CS: CODE, DS: CODE
                                                    RELOC: BYTE, PNTR: WORD, SRCE: BYTE
                                           EXTRN
                                           PUBLIC
                                                    HIDNFL
                                                                               ; Working buffer for hidden file count
                                                    3 DUP (0)
                                  HIDNFL
                                           DB
0000
          03 [
                00
                     ]
                                                                               ; Dummy FCB for file find (extended)
                                  EXTND
                                                    OFFH,0,0,0,0,0
      FF 00 00 00 00 00
                                           DB
0003
                                                                                 File attribute byte
                                  ATRB
                                           DB
      00
0009
                                                    0, '???????????
                                                                               ; Dummy FCB for file find (normal)
         3F
3F
      00
             3F 3F 3F 3F
                                           DB
000A
             3F 3F 3F 3F
       3F
                                           DB
                                                    24 DUP (?)
0016
          18 [
                 22
                     3
                                  ;
                                           PUBLIC
                                                    SCAN
                                                    NEAR
002E
                                   SCAN
                                           PROC
                                                    DI, OFFSET PNTR
                                           MOV
002E
       BF 0000 E
                                            XOR
                                                    AX, AX
0031
       33 CO
0033
                                           MOV
                                                    CX, 120
      89 0078
                                                                               ; Clear pointer table
                                                    STOSM
                                            REP
      F3/ AB
                                                    BX, OFFSET PNTR
                                                                               ; BX to pointer list
                                            MOV
0038
      BB 0000 E
       BF 0000 E
                                                    DI, OFFSET SRCE
                                                                               ; DI to file name stack
                                            MOV
003B
                                                    CX,CX
       33 09
                                            XOR
003E
                                                                               ; Search FCB
                                            MOV
                                                    DX. OFFSET EXTND
       BA 0003 R
0040
                                                                               ; Set for hidden files
                                                    ATRB, 2
                                            MOV
0043
       C6 06 0009 R 02
                                            JMP
                                                     SHORT MAIN
0048
       EB 19
                                                                               ; Set for system files
                                                     ATRB, 4
       C6 06 0009 R 04
                                   SYS:
                                            MOV
004A
                                            JMP
                                                     SHORT MAIN
 004F
       EB 12
                                                                               ; Set for IBMBIO, IBMDOS,...
       C6 06 0009 R 06
                                   IRM:
                                            MOV
                                                     ATRB, 6
0051
                                                     SHORT MAIN
                                            JMP
 0056
       EB OB
                                                                               ; Save hidden file count
                                                     HIDNFL, CL
 0058
       88 OE 0000 R
                                   FIL:
                                            MOV
                                                     CX,CX
                                            XOR
       33 69
 0050
                                                                                ; Set for regular files
       C6 06 0009 R 00
                                            MOV
                                                     ATRB. 0
 005E
                                            MOV
 0063
       B4 11
                                   MAIN:
                                                     AH, 11H
                                                                                ; Get first search entry
                                            INT
                                                     21H
       CD 21
 0065
                                                     SHORT INNER
 0067
                                             JMP
       EB 07
                                                     DX, OFFSET EXTND
       BA 0003 R
                                   LOOP:
                                            MOV
 0069
                                                     AH, 12H
21H
                                             MOV
       B4 12
 0060
                                                                                ; Get next entry
 006E
       CD 21
                                             INT
                                    INNER:
                                             OR
                                                     AL, AL
 0070
       OA CO
                                                     DONE
       75 10
                                             JNZ
 0072
                                             CALL
                                                     SAVE
                                                                                ; File name to stack
       E8 0097 R
A0 0009 R
 0074
                                                                                ; Fetch attribute of search
                                                     AL, ATRB
SI, 93H
AL, BYTE PTR [SI]
 0077
                                             MOV
       BE 0093
3A 04
75 E8
                                                                                ; Point to DTA attribute field
                                             MOV
 007A
                                             CMP
 007B
                                                                                ; Don't count if not requested type
                                                     LOOP
                                             JNE
 007F
                                                                                ; Count entries
 0081
                                             INC
                                                     CX
                                                     LOOP
                                             JMP
 0082
       EB E5
                                                                                ; If zero, all file types checked
       80 3E 0009 R 00
74 0B
                                             CMP
                                    DONE:
                                                     ATRB, 0
 0084
 0089
                                             JZ
                                                      ALLFL
       80 3E 0009 R 04
                                             CMP
                                                     ATRB, 4
 00BB
                                                                                ; =2, next to system
 0090
        72 B8
                                             JB
                                                      SYS
 0092
        74 BD
                                             JE
                                                      IBM
                                                                                ; =4, next to IBM...
                                                                                ; =6, next to regular
        77 C2
                                             JA
                                                      FIL
 0094
        03
                                    ALLFL:
                                             RET
                                                                                ; Returns count in CX
 0096
                                    SCAN
                                             ENDP
 0097
                                    SAVE
                                             PROC
                                                      NEAR
 0097
        80 3E 0009 R 00
75 40
                                                      ATRB, 0
 0097
                                             CMP
                                                      PASS
                                                                                 ; Hidden file count, no name save
  009C
                                             JNZ
  009E
       51
                                             PUSH
                                                      CX
        89 3F
83 C3 02
                                             MOV
                                                      WORD PTR [BX], DI
                                                                                 ; Save pointer to entry stack
  009F
                                                                                     and step pointer table
                                             ADD
  00A1
```

```
PUSH
 00A4
                                                        DI
                                                                                     ; Save address for ordering character
        47
 00A5
                                               INC
                                                        DI
                                                                                     ; Step to start of name
 00A6 BE 0088
                                               MOV
                                                        SI,88H
                                                                                     ; Point to DTA file name
 00A9
        B9 0008
                                               MOV
                                                         CX,8
                                                        AL, BYTE PTR [SI]
 00AC 8A 04
                                      SVLP:
                                               MOV
 00AE 3C 20
                                               CMP
 00B0
        74 06
                                                         NMDNE
                                               17
                                                                                     ; End of name
 0082
        88 05
                                               MOV
                                                         BYTE PTR [DI], AL
 00B4
                                               INC
 00B5
                                               INC
                                                         DI
 0083 47
0086 E2 F4
0088 BE 0090
008B EB 22 90
008E 80 3C 20
00C1 74 09
                                               LOOP
                                                         SVLP
                                                        SI,90H
TESTRE
                                      NMDNE:
                                               MOV
                                                                                     ; Point to DTA type field
                                                JMP
                                                                                     ; Check if relocation requested
                                      STOTYP: CMP
                                                         BYTE PTR [SI], ' '
 00C1
00C3
                                               JZ
                                                         ALLDNE
                                                                                     ; No file type
        C6 05 2E
                                               MOV
                                                         BYTE PTR [DI],'.'
 00C6 47
                                               INC
                                                         DI
        B9 0003
 00C7
                                                        CX.3
                                               MOV
 00CA F3/ A4
00CC C6 05 00
                                               REP
                                                         MOVSB
                                                                                    ; Move type field to stack
; Mark end of string
 00CC C6
00CF 47
                                     ALLDNE: MOV
                                                        BYTE PTR [DI],0
                                               INC
                                                         DI
 00D0 BE 00A4
                                               MOV
                                                        SI, OA4H
                                                                                     ; Point to file size
 00D3 B9 0004
                                               MOV
                                                         CX,4
 00D6 F3/ A4
                                               REP
                                                        MOVSB
                                                                                         and save in stack
                                                                                    ; 'Ordering' character
; Location for same (PUSH from DI)
; Put at start of string
 0008
                                               POP
                                                        AX
        50
 0009
                                               POP
 00DA 88 46 00
00DD 59
00DE C3
                                               MOV
                                                        [BP],AL
                                               POP
                                     PASS:
                                               RET
 OODF
        57
                                      TESTRE: PUSH
                                                        DI
                                                        AL,OAOH
DI,OFFSET RELOC
BYTE PTR [DI],O
 00E0 B0 A0
                                               MOV
                                                                                    ; Default order high
 00E2 BF 0000 E
                                               MOV
 00E5 80 3D 00
                                     TLP:
                                               CMP
                                                                                    ; End of reorder table
00E8 74 13
00EA 57
00EB 56
                                               JE
                                                        NOMTCH
                                               PUSH
                                                        DI
                                                                                     ; Save start of string in table
                                               PUSH
                                                        SI
                                                                                           and FCB location pointer
 00EC B9 0003
                                                        CX.3
                                               MOV
00EF F3/A6
00F1 5E
00F2 5F
                                               REPE
                                                        CMPSB
                                                                                     ; File extension to table
                                               PNP
                                                        SI
                                               POP
                                                        DI
00F3 74 05
                                               JE
                                                        MATCH
00F5 83 C7 04
00F8 EB EB
                                                        DI.4
                                               ADD
                                                                                     ; Point to next entry
                                                        TLP
                                               JMP
00FA 8A 45 03
                                     MATCH:
                                               MOV
                                                        AL,[DI+3]
                                                                                    ; Reorder character to AL
00FD 5F
                                     NOMTCH: POP
                                                        DI
                                                                                    ; Restore to value at entry
OOFE 50
OOFF EB BD
                                               PUSH
                                                        AX
                                                                                    ; Reorder char to stack
                                               JMP
                                                        STOTYP
0101
                                     SAVE
                                               ENDP
0101
                                     CODE
                                               FNDS
                                               END
Segments and groups:
                   Name
                                                Size
                                                         alion
                                                                 combine class
0101
                                                         PARA
                                                                   PUBLIC 'CODE'
Symbols:
                   Name
                                               Type
                                                         Value
                                                                  Attr
ALLDNE
ALLFL.
ATRB
DONE
EXTND.
FIL.
HIDNFL
IBM.
INNER.
LOOP
MAIN
MATCH.
NMDNE.
NOMTCH
                                               L NEAR
                                                         0000
                                               L NEAR
                                                         0096
                                                                  CODE
                                                         0009
                                               L BYTE
                                                                   CODE
                                               L NEAR
                                                         0084
                                                                  CODE
                                               L BYTE
                                                         0003
                                                                   CODE
                                               L NEAR
L BYTE
                                                         0058
                                                                   CODE
                                                         0000
                                                                   CODE
                                                                            Global Length =0003
                                               L NEAR
                                                         0051
                                                                   CODE
                                               L NEAR
                                                         0070
                                                                   CODE
                                                                  CODE
                                               L NEAR
                                                         0069
                                                                   CODE
                                               L NEAR
                                                         0063
                                                         OOFA
                                               L NEAR
                                                                   CODE
                                               L NEAR
                                                         0088
                                                                   CODE
                                               L NEAR
                                                         OOFD
                                                                  CODE
```

UNPARALLELED

and Porta

in an ISAM PAGK at am UNBE

1979 and created ACCESS MANAGER™ for Digital Research, now redefines the market for high performance, B-Tree based file handlers. With c-tree™

- complete C source code written to K & R standards of portability
- high level, multi-key ISAM routines and low level B-Tree functions

The company that introduced micros to B-Trees in

- routines that work with single-user and network systems
- no royalties on application programs

\$395 COMPLETE

Specify format: 8" CP/M® 51/4" PC-DOS 8" RT-11

for VISA, MC or COD orders, call toll free 1-800-232-3344

Access Manager and CP/M are trademarks of Digital Research, Inc. c-tree is a trademark of FairCom.

© 1984 FairCom

2606 Johnson Drive Columbia MO 65203

Circle no. 22 on reader service card.

GANGPRO-8™ \$995

RS-232 STAND-ALONE, INTELLIGENT, EASY TO USE

Program & Verify ALL 5V 25 & 27 series EPROMS & ALL Micros: 8748/49/41/42/51/55 68701,68705,38E70

HIGH **THROUGHPUT** GANG PROGRAMMING 4, 8, 12, 24, EPROMS at a time

UV ERASERS



LOGICAL DEVICES, INC. 1-800-EE1-PROM

Florida 305-974-0967

Each generation dropped the price by \$1000 PROMPRO-8 Equivalent Price 5 Years ago: \$5689

PRICE TODAY: \$689!!

14 DAY MONEY BACK GUARANTEE

PROMPRO-8



- Buffer (64 K, STD)
- * EPROM Simulation
- * Terminal/Computer Mode
- * Download/Upload Hex Files

PROMPRO-7™ \$489

* 32K RAM Buffer

PRICES FROM: \$49.95

> ERASE 30 EPROMS AT A TIME!

PAL PROGRAMMERS™

PALPRO-1™\$ 599 PALPRO-2™ \$1195

Circle no. 34 on reader service card.

Letters (Listing continued, text begins on page 8) **Listing Three**

PASS PATE PASS PATE PASS PASS PASS PASS PASS PASS PASS PAS	 	 	 	 	 	 	VNNVLLL	NEAR WORTE PROCC PROTE PROTE PROCC PROCC NEAR NEAR NEAR NEAR	00DE 0000 0000 0097 002E 0000 00BE 00AC 00DF 00E5	00 00 00 00 00 00 00 00	00E 10E 10E 10E 10E 10E 10E 10E 10E	External External Length =006A Global Length =0069 External
Warning Errors 0												

End Listing Three

Listing Four

```
,132
                                             PAGE
                                             TITLE
                                                      COPRNT
                                                              - Diskette contents list - Print Cover Sheet
                                             COMMENT
                                                               * Version 1.0 - June 1983
                                                                 Modified Jan 1984 *
                                             PAGE
                                                      82
                                   CODE
0000
                                             SEGMENT PARA PUBLIC 'CODE'
                                             ASSUME CS:CODE, DS:CODE
                                                     PNTR: WORD, STKCNT: WORD, TITLX: BYTE, HIDNFL: BYTE
                                             EXTRN
                                             EXTRN
                                                     CONVRT: NEAR, DECMAL: NEAR
                                            PUBLIC
                                                     PSX, RESTR
0000 00
                                   PSX
                                             DB
                                                      0
                                                                                 ; Pass counter
0001 OC 00
                                   RESTR
                                                      12.0
                                            DR
                                                                                 ; Printer 'restore' forms
; Work buffer for file size
0003
          07 [
                                   DBUF
                                             DR
                                                      7 DUP (0)
                 00
000A
       48 69 64 64 65 6E
                                   HIDF
                                            DR
                                                     'Hidden files '.0
       20 66 69 60 65 73
      20 20 00 20 20 00
                                                     ;;;;0
0019
                                   DBLK
                                            DB
                                                                                 ; Double blank between columns
      7C 20 20 00
20 20 7C
0010
                                   LFTB
                                            DB
                                                                                 ; Left border
; Right border (include CR/LF)
0020
                                   RGTB
                                            DB
      OD 0A 00
0023
                                   CRLF
                                            DB
                                                     13,10,0
0026
      00
                                   BCNT
                                            DR
                                                                                 ; Body line counter
                                            PUBLIC
                                                     PRINT
0027
                                   PRINT
                                            PROC
                                                     NEAR
0027
      FE 06 0000 R
                                            INC
                                                     PSX
                                                                                 ; Count number of prints
002B C6 06 0026 R 21
                                            MOV
                                                     BCNT, 33
                                                                                 ; Set body line counter
0030 A1 0000 E
                                            MOV
                                                     AX, STKCNT
                                                                                 ; Load entry count
0033 B6 04
0035 F6 F6
                                            MOV
                                                     DH, 4
                                            DIV
                                                     DH
                                                                                 ; Divide by number of columns
0037 0A E4
0039 74 03
                                            OR
                                                     AH, AH
                                            JZ
                                                     SETCHT
                                                                                 ; Evenly divisible
003B
003D
      FE CO
                                            INC
                                                                                 ; Ragged edge
                                            CRW
003E
                                   SETCHT: PUSH
                                                                                 ; Entries per column count
      E8 014D R
E8 013E R
E8 015C R
003F
                                                     DOBRDR
                                            CALL
                                                                                 ; Do upper border
0042
                                            CALL
                                                     DOBLNE
                                                                                    and a blank line
0045
                                            CALL
                                                     DOLFTM
                                                                                 ; Do left margin
0048
      BA 0000 E
                                            MOV
                                                     DX, OFFSET TITLX
      E8 00A5 R
                                            CALL
                                                     DOPRT
                                                                                 ; Output the title line
004E
      B9 0004
                                                     CX.4
CLER
                                            MOV
0051
      E8 016E R
                                            CALL
                                                                                 ; Add 4 spaces
      E8 0165 R
E8 013E R
0054
                                            CALL
                                                     DORGTM
                                                                                 ; Do right margin
0057
                                            CALL
                                                     DOBLNE
                                                                                     and another blank line
005A
                                            POP
```

(Continued on page 18)

Letters (Listing continued, text begins on page 8)

Listing Four

```
BP,CX
BP,1
SI,OFFSET PNTR
005B 8B E9
                                              MOV
                                                                                   ; Offset per column in pointer list
005D D1 E5
                                              SHL
                                                                                   ; Point to start of pointer list
                                              MOV
005F
      BE 0000 E
0062 E8 015C R
                                    OTLP:
                                              CALL
                                                       DOLFTM
                                                                                   ; Do a left margin
                                                                                   ; Set inner loop count to column # ; Clear column offset reg
      B2 04
33 DB
E8 00B9 R
                                                       DL,4
BX,BX
                                              MOV
0065
0067
                                              XOR
0069
                                              CALL
                                                       PRTENT
                                                                                   ; Print stack entry
                                    INLP:
006C 03 DD
                                                                                   ; Step to next column entry
                                              ADD
                                                       BX, BP
                                                       DL
006E FE CA
                                              DEC
                                                       INLP
                                                                                   ; End of inner loop
0070 75 F7
                                              JNZ
0072 E8 0165 R
0075 83 C6 02
0078 FE 0E 0026 R
007C E2 E4
007E 32 C0
                                                                                   ; Do a right margin
                                              CALL
                                                       DORGTM
                                                       SI.2
                                                                                   ; Step to next pointer
                                              ADD
                                                       BCNT
                                                                                   ; Decrement body line count
                                              DEC
                                                                                   ; End of outer loop
                                              LOOP
                                                       OTLP
                                                       AL, AL
AL, HIDNFL
                                              XOR
0080 0A 06 0000 E
0084 74 08
                                                                                   : Check for any hidden files
                                              JZ
                                                       BLNK
                                                                                   ; Print hidden files count,
                                              CALL
                                                       PRHID
0086 E8 0108 R
                                                       BCNT, 2
CL, BCNT
                                                                                       uses two lines
0089 80 2E 0026 R 02
008E 8A 0E 0026 R
                                              SUB
                                                                                   : Load remaining body lines
                                              MOV
                                    BLNK:
                                                        CH, CH
0092
       32 ED
                                              XOR
                                                                                   ; All used ; Fill out body lines
                                                       NOFILL
                                              JCXZ
0094 E3 05
0096 E8 013E R
0099 E2 FB
                                     FILL:
                                              CALL
                                                        DOBLNE
                                              LOOP
                                                        FILL
                                                                                    ; Do bottom border
                                                        DOBRDR
009B E8 014D R
                                     NOFILL: CALL
                                                        DX, OFFSET RESTR
009E BA 0001 R
                                              MOV
                                              CALL
                                                        DOPRT
                                                                                    ; Restore page
 00A1 E8 00A5 R
                                              RET
 00A4
                                              ENDP
                                     PRINT
 00A5
                                              PUBLIC
                                                       DOPRT
                                                                                    ; This subroutine prints string
                                     DOPRT
                                                        NEAR
                                              PROC
 00A5
                                                                                       (pointer in DX on entry).
                                              PUSH
                                                        DX
       52
 00A5
                                                                                    ; String terminated by a null byte
                                               PUSH
                                                        SI
 00A6
                                                        SI,DX
                                               MOV
       8B F2
 00A7
                                                        AH,5
DL,BYTE PTR [SI]
                                                                                    ; Print line function
                                               MOV
 00A9 B4 05
                                     DPLP:
                                               MOV
 00AB 8A 14
                                                        DL.DL
                                               OR
 OOAD
        0A D2
                                                        PRTEND
 OOAF
       74 05
                                               JZ
                                               INT
                                                        21H
 00B1
       CD 21
       46
EB F5
                                               INC
 00B3
                                               JMP
                                                        DPLP
 00B4
                                              POP
                                     PRTEND:
                                                        SI
 0086
        5E
 00B7
                                               POP
                                                        DX
                                               RET
 8800
 0089
                                      DOPRT
                                               ENDP
                                                                                    ; Print one stack entry
                                     PRTENT
                                               PROC
                                                        NEAR
 0089
 0089 51
008A 52
008B 89 000C
                                               PUSH
                                                         CX
                                               PUSH
                                                        DX
                                                        ČX,12
DI,WORD PTR [SI+BX]
DI,DI
                                               MOV
 00BE 8B 38
00C0 0B FF
00C2 74 37
                                                                                    ; DI points to stack entry
                                               MOV
                                                                                     ; If entry is zero, blank space
                                               OR
 00C2
00C4
                                               17
                                                         BLNK1
                                               INC
                                                                                     ; Step over reorder character
       47
                                                         DI
 00C5
00C7
       B4 05
                                               MOV
        8A 15
                                      PELP:
                                               MOV
                                                         DL, BYTE PTR [DI]
                                                                                     ; Print to the end of the
                                                                                        name/type entry
 0009 0A D2
                                               OR
                                                         DL, DL
 00CB 74 36
                                                                                        blank rest of 12 chars
                                               JI
                                                         BLNK2
                                                         21H
        CD 21
 0000
                                               INT
  OOCF
        47
                                               INC
                                                         DI
                                                         PELP
 0000
        E2 F5
                                               LOOP
       B9 0001
  0002
                                      BACK:
                                               MOV
                                                         CX.1
                                                         CLER
                                                                                     ; Insert 1 space
  0005
        E8 016E R
                                               CALL
                                                INC
  8000
                                                         AX, WORD PTR [DI]
DX, WORD PTR [DI+2]
  00D9 8B 05
                                                MOV
                                                                                     ; Load file size
  00DB 8B 55 02
00DE 56
                                                MOV
                                                PUSH
                                                         SI
                                                         DI, OFFSET DBUF
  OODF
        BF 0003 R
                                                MOV
        E8 0000 E
                                                CALL
                                                         CONVRT
                                                                                     ; Size to ASCII
  00E2
                                               POP
  00E5
                                                         DX, OFFSET DBUF
  00E6
         BA 0003 R
                                                MOV
                                                                                     ; Print the size
  00E9
         E8 00A5 R
                                                CALL
                                                         DOPRT
                                      GONE:
                                                POP
  OOEC
```

```
OOED
                                            PUSH
                                                                               ; Reload entry value
OOEE FE CA
                                           DEC
                                                    DL
00F0
       74 06
                                            JI
                                                    PUNT
                                                                               ; If last column don't space over
00F2 BA 0019 R
00F5 EB 00A5 R
                                            MOV
                                                    DX, OFFSET DBLK
                                            CALL
                                                    DOPRT
                                                                               ; Print 2 blanks between cols
00F8
                                  PUNT:
                                            POP
                                                    DX
00F9
                                            POP
                                                    CX
00FA
       0.3
                                            RET
OOFB
      B9 0013
                                   BLNK1:
                                           MOV
                                                    CX, 19
                                                                               ; No entry, blank entire column
     E8 016E R
EB E9
OOFE
                                                    CLER
                                            CALL
0101
                                            JMP
                                                    GONE
     E8 016E R
0103
                                   BLNK2:
                                           CALL
                                                    CLER
                                                                               ; Blank remainder of field
0106
      ER CA
                                            JMP
                                                    BACK
0108
                                   PRTENT
                                           ENDP
0108
                                   PRHID
                                           PROC
                                                    NEAR
0108 E8 013E R
                                           CALL
                                                    DOBLNE
                                                                               ; Blank line
0108
       E8 015C R
                                            CALL
                                                    DOLFTM
                                                                               ; Left margin
      B9 0014
010E
                                           MOV
                                                    CX.20
                                                                               ; Twenty blanks
0111
      E8 016E R
                                            CALL
                                                    CLER
0114 BA 000A R
                                           MOV
                                                    DX, OFFSET HIDF
0117
      E8 00A5 R
                                            CALL
                                                    DOPRT
                                                                               ; Text
011A
      A0 0000 E
                                                    AL, HIDNFL
DI, OFFSET HIDNFL
                                           MOV
011D BF 0000 E
                                            MOV
0120 8B D7
                                           MOV
                                                    DX.DI
                                                                               ; Set address for print (below)
0122
      E8 0000 E
                                            CALL
                                                    DECMAL
                                                                               ; Number of files to ASCII
0125
      80 3E 0000 E 30
                                           CMP
                                                    HIDNFL, '0'
012A
      75 05
                                            JNE
                                                    TWOCH
012C C6 06 0000 E 20
                                           MOV
                                                    HIDNFL, ' '
                                                                               ; Zero suppress
0131 E8 00A5 R
                                   TWOCH:
                                           CALL
                                                    DOPRT
0134
0137
      B9 002E
                                           MOV
                                                    CX,46
      E8 016E R
                                           CALL
                                                    CLER
                                                                               ; Fill with blanks
013A E8 0165 R
                                           CALL
                                                    DORGTM
                                                                               ; Right margin
013D C3
                                           RET
013E
                                  PRHID
                                           ENDP
013E
                                  DOBLNE
                                           PROC
                                                    NEAR
013E
      51
                                           PUSH
                                                    CX
013F
0142
      E8 0150 R
B9 0052
                                           CALL
                                                    DOLFTM
                                                                               ; Output a bordered blank line
                                                    CX.82
CLER
                                           MOV
0145
      E8 016E R
                                           CALL
0148
      E8 0165 R
                                           CALL
                                                    DORGTM
014B
      59
                                           POP
                                                    CX
0140
      03
                                           RET
014D
                                  DOBLNE
                                          ENDP
0140
                                  DOBRDR
                                           PROC
                                                    NEAR
      B9 0058
0140
                                                    CX,88
DL,'-'
                                           MOV
0150
      B2 2D
                                           MOV
0152
0155
      E8 0170 R
                                                    DLFIL
                                           CALL
                                                                               ; Output a top or bottom border
      BA 0023
                                           MOV
                                                    DX. OFFSET CRLF
0158
      E8 00A5 R
                                           CALL
                                                    DOPRT
015B
                                           RET
0150
                                  DOBRDR
                                           ENDP
0150
                                  DOLFTM
                                           PROC
                                                    NEAR
                                                                              ; Output ";
0150
      52
                                           PUSH
                                                    DX
0150
      BA OOIC R
                                           MOV
                                                    DX, OFFSET LFTB
0160
      E8 00A5 R
                                           CALL
                                                    DOPRT
0163
0164
                                           POP
                                                    DX
      03
                                           RET
0165
                                  DOLFTM
                                           ENDP
                                  DORGTM
0165
                                           PROC
                                                    NEAR
                                                                                              I H
                                                                              ; Output "
0165
      52
                                           PUSH
                                                    DX
      BA 0020 R
0166
                                           MOV
                                                    DX, OFFSET RGTB
0169
      E8 00A5 R
                                           CALL
                                                    DOPRT
      5A
0160
                                           POP
                                                    DX
0160
      C3
                                           RET
016E
                                  DORGTM
                                           ENDP
016E
                                  CLER
                                           PROC
                                                    NEAR
                                                                              ; Output CX blanks
                                                    DL,
016E
      B2 20
                                           MOV
      B4 05
0170
                                  DLFIL:
                                           MOY
                                                                              ; Output DL character CX times
      CD 21
E2 FC
0172
                                  CLRLP:
                                           INT
                                                    21H
0174
                                           LOOP
                                                    CLRLP
0176
                                           RET
```

Letters (Listing continued, text begins on page 8) **Listing Four**

0177	CLER	ENDP			
0177	CODE	ENDS			
	,	END			
Segments and groups:					
Name		Size	align	combine	class
		0177	PARA		'CODE'
CODE		VIII	i niin	ODETO	CODE
Symbols:				A1.1	
Name		Туре	Value	Attr	
BACK BCNT BLNK BLNK BLNK BLNK1 BLNK2 CCLER CCLRLP CCNVRT CRLF DBLK DBUF DECMAL DLFIL DOBLNE DOBRDR DOUFIM DOPRT DORGTM DPLP FILL GONE HIDF HIDNFL INLP LFTB NOFILL OTLP PELP PNTR PRHID PRITENT PRESTR RGTB SETCNT STKCNT TIVIX TWOCH.		ARE AREARCE AREA AREA AREA AREA AREA ARE	0108 0027 0086 0089 0000 0078 0001 0020 003E 0000	CODE CODE CODE CODE CODE CODE CODE CODE	Length =0009 External Length =0007 External Length =000F Length =0009 Global Length =0014 Length =0009 External External Length =0036 Global Length =007E Length =004F Global Global External External External External

End Listing Four

Listing Five

```
.132
COENDP - Diskette contents list - Workarea Definitions
* Version 1.0 - June 1983
Modified - Feb 1984 *
                                               PAGE
                                               TITLE
                                               COMMENT
                                               PAGE
                                     CODE
                                              SEGMENT PARA PUBLIC 'CODE'
ASSUME CS:CODE, DS:CODE
0000
                                               PUBLIC SRCE, PNTR, RELOC
= 0000
                                     RELOC
                                               EQU
                                                        THIS BYTE
                                                                                     ; File extension reorder list
                                     PNTR
= 0029
                                               EQU
                                                        RELOC+10*4+1
                                                                                     ; Pointer list
= 0119
                                               EQU
                                                        PNTR+240
                                                                                     ; Start of entry stack
                                     CODE
0000
                                               ENDS
                                               END
Segments and groups:
                   Name
                                               Size
                                                         align
                                                                  combine class
CODE . . . .
                                               0000
                                                         PARA
                                                                  PUBLIC 'CODE'
Symbols:
                   Name
                                               Type
                                                         Value
                                                                  Attr
E BYTE 0029
E BYTE 0000
E BYTE 0119
                                                                  CODE
                                                                            Global
                                                                  CODE
                                                                            Global
                                                                  CODE
                                                                            Global
Warning Severe
Errors Errors
0 0
```

1 1 1		IBM PC-	DOS MASTER DIS	KETTE		Free:	7168	02/19/	84
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	BASIC.COM BASICA.COM CHKDSK.COM COMMAND.COM COMP.COM DEBUG.COM DISKCOMP.COM	11392 16768 1720 4959 1649 5999 1640	DISKCOPY.COM EDLIN.COM FORMAT.COM MODE.COM SYS.COM ART.BAS BALL.BAS	2008 2392 3816 2509 605 1920 2048	CALENDAR. BAS CIRCLE. BAS COLORBAR. BAS COMM. BAS DONKEY. BAS MORTGAGE. BAS MUSIC. BAS	3840 1664 1536 4352 3584 6272 8704		IN.EXE	2304 2432 1920 1280 41856
,			Hidden files	2					

DR. DOBB'S CLINIC

by D.E. Cortesi, Resident Intern

Transcendental Sources

Some time back we published Richard Falk's request for sources of good algorithms for the transcendental functions. Several people responded with suggestions, notably R. C. Briggs, Thomas Chrapkiewicz, Tom Enterline, David Feign, Calvin L. Gardner, Glenn F. Roberts, and William A. Rutiser. Here are the titles they suggested that Falk read, with our comments on those we've used:

Approximations For Digital Computers, by Cecil Hastings, Jr., Princeton University Press, 1955. A very concise introduction to Chebyshev approximation methods, plus seventy-odd approximation formulae with graphs of their error functions.

BASIC Scientific Subroutines, Volumes I and II, by F. R. Ruckdeschel, McGraw-Hill, 1981.

Computer Approximations, by John F. Hart, et al., Robert E. Krieger Publishing Company, Krieger Drive, Malabar, FL 32950, 1968. Mentioned by two respondents. Rutiser sent a copy of the table of contents, which looks very complete.

Handbook of Mathematical Functions, edited by Milton Abramowitz and Irene A. Stegun, National Bureau of Standards Applied Mathematics Series #55, U.S. Government Printing Office, Washington, DC 20402.

Numbers In Theory and Practice, edited by Blaise W. Liffick, Byte Books. A rather miscellaneous collection of early BYTE articles, this is a useful introduction to floating-point arithmetic and to random number generation but not a complete or accurate source book.

Scientific Analysis on the Pocket Cal-

culator, by Jon M. Smith, John Wiley and Sons, 1975. A friendly introduction to numerical analysis written for "those who owned or operated a modern electronic pocket or desk calculator" just as the age of the personal computer was dawning. Many approximation formulae with tables of error functions.

Software Manual for the Elementary Functions, by William J. Cody, Jr., and William Waite, Prentice-Hall, 1980. The book most often cited by our respondents and the one that comes closest to being a true cookbook, this is the reference of choice for Falk and anyone else who wants to write math functions in assembly language. Sixteen transcendental functions are presented as flowcharts for programs that can deal with float numbers at the bit or digit level; that is, at the level where the exponent and the fraction can be treated as separate integers. Where necessary, different algorithms are given for binary and decimal (BCD) methods. A great deal of attention is paid to accuracy. The function vocabulary is taken from Fortran, and some familiarity with that language is a help initially in getting into the book.

Over-Submitting

We're longtime advocates of using the batch facilities of one's operating system—SUBMIT and XSUB in CP/M 2.2, BAT files in MS-DOS, etc. Steve Grant of Hacienda Heights, CA, has the same bent, but he overloaded the XSUB facility of CP/M 2.2. Here's what he says:

"How do you do an XSUB file longer than 128 lines? Garden-variety SUB-MIT files are easy since the last line of one file can just SUBMIT the next one. But that won't work with XSUB since the second SUBMIT command will be presented, not to the CCP, but to some innocent program that has never heard of the command line 'submit foo.'

"I got into this mess while trying to make my machine an Adventure Grandmaster. I carefully created a file as follows:

> xsub adv (start the Adventure program)

> ... (lots and lots of commands
> ... to move around the cave,
> etc.)

I named it TROLL.SUB and did a SUB-MIT TROLL.

"Disaster! It started executing well past the beginning without ever executing ADV. The CCP was thoroughly confused by the command GET SPICE. Upon closer examination it became clear the TROLL would always start executing 128 lines from the end. I threw out all but the first 128 lines. Now the program plays the first 126 moves just fine then stops and waits for human activity. Any ideas?"

Unfortunately, no. The problem lies in the peculiar format into which SUB-MIT processes a submitted file and in the system's belief that a processed file won't exceed one logical extent of 16K (128 times 128). We could suggest that Steve convert to CP/M Plus or to MS-DOS version 2, both of which allow console input to be redirected to an ASCII file of any length, but we doubt that he'd find that constructive.

Or maybe he could find a way to make the Adventure program take multiple commands in one physical line? Reader suggestions are welcome....

An Old Bug

Frequent correspondent David McLanahan got bit by an old bug. In fact, it's so old, and has been so faithfully transported to system after system, it might be useful to print his warning.

"I've created a number of .BAS files with lower-case names from MBASIC. They are a big bother; the directory lists them, but CP/M can't find, erase, or rename them. [You can use BASIC's KILL command to delete them.—DEC] A friend was over the other day and saved a program to disk in ASCII form. My friend meant to enter

SAVE "PROG", A

but by accident typed it as

SAVE "PROG,A

The final quote is optional—when nothing follows the filename! The file was created as "PROG,A.BAS" and not in ASCII form. From there it got into MAST.CAT. As you undoubtedly know, the catalog programs use commas as the delimiters between entries, thus the catalog got out of sync and blew 20K worth of listings! Mutter, mutter..."

Do the newer Microsoft BASICs do this? Somebody check for us on a PC with DOS 2.1 and on a Macintosh, please.

[Editor's Note: Upon reading this, we decided to try this ourselves on the IBM PC here in our office. Mr. McLanahan's problem has apparently been fixed on our BASICA for PC DOS 2.1. When we typed SAVE "PROG, A and hit return our request was refused and we received a somewhat cryptic "too many files" message. This, incidentally, is the same message we receive when asking BASICA to save a file with a filename containing over 10 characters plus the .BAS extension. Anyone know why this message is generated as opposed to one of the others that indicates a bad filename? — RW]

BASIC Precision, Again

Several readers were bothered by our recent discussions of BASIC single-and double-precision numbers and the ills they are heir to. Bill Metzenthen, David S. Tilton, William R. Hamblen, and David Feign all wrote to make essentially the same point: namely, that an understanding of how binary float numbers are represented is essential to an understanding of how errors are formed and propagated. Let Tilton tell

it, since he has a positive suggestion.

"What most people do not realize is that, when they talk about floatingpoint numbers in a computer, it is a binary point and not a decimal point that is floating around inside the machine.

"Everybody knows that there are some fractions that never come out even to any number of places. Thirds and sevenths, for example, cannot be expressed exactly as decimals. In the binary system even fewer fractions come out even. Usually these arise in money amounts where cents are expressed in hundredths of dollars. The only hundredths that will come out even as binary fractions are .00, .25, .50, and .75. All the others will have some small roundoff error no matter how many binary places (i.e., bits) are allotted to represent them.

"Fortunately there is a simple solution to the problem: store money amounts internally as an integral number of pennies. Not with the type integer, which would limit you to a maximum of \$327.68. Single precision can devote 24 bits to an integer, and double precision can devote 56 bits. This is roughly equivalent to 7.2 and 16.8 decimal places, respectively. The only times you need to worry about round-off errors are when you multiply by 100 on input and divide by 100 on output."

For the nit-pickers out there, we should qualify Tilton's statement about the fractions that can or can't be represented accurately. The fractions .00, .25, .50, and .75 are the only *two*-

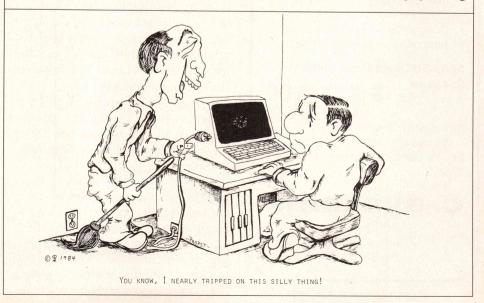
digit decimal ratios that can be represented exactly in binary. If you divide a quarter by two, the result is a complete and accurate representation of 0.125: twelve-and-a-half cents.

His solution (carry currency amounts as integral numbers of pennies) and his point that you can use floating-point variables to contain large integral values are both useful suggestions. There are pitfalls here as well, however. For example, if you multiply an integral number of pennies by our local sales tax rate of 0.065, you will have a fractional part in your result. The fraction represents not pennies but mills; nevertheless, it is subject to all the same problems of roundoff. Input can be a problem as well. Multiplying an input amount by 100 isn't sufficient; the input conversion routine might already have done some damage before you apply the scale factor.

Public Opinion Poll

Peter Baenziger of Kalamazoo, MI, takes us to task for the diatribe on polling loops we tacked onto the May column. Here's his counter-blast.

"You take Michael Barr to task for writing a polling loop checking the Caps Lock status on an IBM PC. You state, 'This is an example of how a solution can go wrong when it is implemented at the wrong level of the system.' You are probably right; it isn't the right level. But I don't think Barr and his loop are wrong. In my opinion, if the loop causes problems, it's a weakness in the multitasking operating



Software Development PCDOS/MSDOS

Complete C Compiler

- Full C per K&R
- Inline 8087 or Assembler Floating Point, Auto Select of 8087
- Full 1Mb Addressing for Code or Data
- Transcendental Functions
- ROMable Code
- Register Variables
- Supports Inline Assembler Code

MSDOS 1.1/2.0 Library Support

- All functions from K&R
- All DOS 2.0 Functions
- Auto Select of 1.1 or 2.0
- Program Chaining Using Exec
- Environment Available to Main

c-window™ Symbolic Debugger

- Source Code Display
- Variable Display & Alteration Using C Expressions
- Automatic Commands
- Multiple Breakpoints by Function & Line Number

8088/8086 Assembler

- FAST Up to 4 times Faster than IBM Assembler
- Standard Intel Mnemonics
- Compatible with MSDOS Linker
- Supports Full Memory Model

8088 Software Development Package

Includes: C Compiler/Library, c-window, and Assembler, plus Source Code for c-systems Print Utility

c-systems

P.O. Box 3253 Fullerton, CA 92634 714-637-5362

Circle no. 15 on reader service card.

system.

"Concurrent CP/M, for example, is sold for the IBM PC, and as such, supposedly supporting a specific computer, it should support the idiosyncracies, the strengths and weaknesses, of the hardware.

"We're living in an imperfect world. The imperfect computers are out there by the hundreds of thousands. An extremely small minority run Concurrent CP/M or Windows, etc. It's ludicrous, in my opinion, that a vast majority of users, millions of people, should always have to take a first peck at the keyboard to find out what the Caps Lock status is just so that a rare multitasking OS won't get into trouble.

"If the multitasking OS is properly designed, it will have saved the state of the machine, including the shift status, before relegating a task to the background. When the task moves to the foreground, the OS should restore the old shift status in 40:17, the polling loop will get it, and all will be well.

"Let me reiterate: I agree with you that two LEDs would be preferable. Or if not LEDs, then a proper DOS call. I even agree that some polling loops are troublesome in a multitasking environment. But the fact is, the two ideal solutions don't exist. Complaining that they should doesn't change reality. So I wouldn't call a program that brings a solution for *most* users a 'bad citizen.' Maybe what we need is better government."

Well, it's a thorny thicket of problems, and Baenziger has done us all a favor by pointing that out. There are lots of angles to explore. An important one, as Baenziger says, is the fact that all those PCs and PC clones exist, and most of them run (and will continue to run) plain old MS-DOS versions 1.2 or 2.0—single-task systems in which a polling loop won't cause any harm at all. Ditto for the near-million CP/M 2.2 systems and the near-million Apple DOSes.

Yet another angle is the culture from which the programmers come. On big systems, programmers tend to take the operating system interface as a given; they play by rules. People writing applications for an IBM 370 wouldn't dream of writing a polling loop. They know that their programs

will be sharing the machine with many others; that the system as a whole will run more efficiently if their programs go to sleep as quickly as possible when they have nothing to do; that the less CPU time they consume, the higher their programs' priorities will become; and that every unnecessary machine cycle will be reflected in the user's timesharing bill.

None of these things are part of the gestalt knowledge associated with personal computers. Multitasking concepts just aren't in the vocabulary of the average grass-roots programmer.

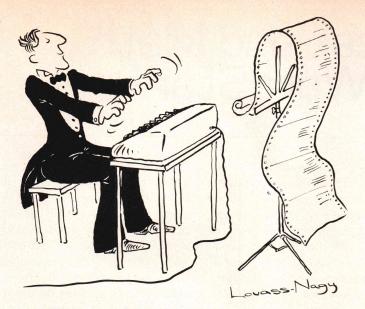
Had IBM foreseen the coming of multitasking to the PC, they could have done something about that. For instance, they could have added a level of indirection to all those low-storage values (i.e., "For keyboard status, poll the byte whose address is at 40:17; to update the screen image, poke a byte based on the address at xyz."). That would have allowed an OS to set aside a machine-status block for each task, swapping the relevant pointers into low storage before activating a task.

The PC's success often makes us overlook the colossal marketing blunder that IBM almost made. If you look at the design, specs, and documentation of the PC circa 1982, you will realize that IBM intended the PC to be an up-market Apple. That's the framework for its BIOS design: one user, one program in BASIC (loaded from cassette), lots of peeks and pokes, fun and games, whee! Thanks largely to independent vendors, the machine turns out to have enough horsepower to be a lot more, but it is still saddled with that simple-minded internal design.

As a result, you get people like Yr. Intern who say, "Obviously a program should be written for good multitasking performance," and also people who say, "Obviously my program ought to use whatever it needs to make it work well."

Who's right? Both? Neither . . . ?

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service No. 191.



·NEW PRODUCTS·

Before Johann Sebastian Bach developed a new method of tuning, you had to change instruments practically every time you wanted to change keys. Very difficult.

Before Avocet introduced its family of cross-assemblers, developing micro-processor software was much the same. You needed a separate development system for practically every type of processor. Very difficult and very expensive.

But with Avocet's cross-assemblers, a single computer can develop software for virtually any microprocessor! Does that put us in a league with Bach? You decide.

The Well-Tempered Cross-Assembler

Development Tools That Work

Avocet cross-assemblers are fast, reliable and user-proven in over 3 years of actual use. Ask NASA, IBM, XEROX or the hundreds of other organizations that use them. Every time you see a new microprocessorbased product, there's a good chance it was developed with Avocet cross-assemblers.

Avocet cross-assemblers are easy to use. They run on any computer with ${\sf CP/M}^*$ and process assembly language for the most popular microprocessor families.

5½" disk formats available at no extra cost include Osborne, Xerox, H-P, IBM PC, Kaypro, North Star, Zenith, Televideo, Otrona, DEC.

Turn Your Computer Into A Complete Development System

Of course, there's more. Avocet has the tools you need from start to finish to enter, assemble and test your software and finally cast it in EPROM:

Text Editor VEDIT -- full-screen text editor by CompuView. Makes source code entry a snap. Full-screen text editing, plus TECO-like macro facility for repetitive tasks. Pre-configured for over 40 terminals and personal computers as well as in userconfigurable form.

CP/M-80 version	\$150
CP/M-86 or MDOS version	\$195
(when ordered with any Avocet pro	duct)

EPROM Programmer -- Model 7128 EPROM Programmer by GTek programs most EPROMS without the need for personality modules. Self-contained power supply ... accepts ASCII commands and data from any computer through RS 232 serial interface. Cross-assembler hex object files can be down-loaded directly. Commands include verify and read, as well as partial programming.

PROM types supported: 2508, 2758, 2516, 2716, 2532, 2732, 2732A, 27C32, MCM8766, 2564, 2764, 27C64, 27128, 8748, 8741, 8749, 8742, 8751, 8755, plus Seeq and Xicor EEPROMS.

Avocet Cross-assembler	Target Microprocessor	CP/M-80 Version	• CP/M-86 IBM PC, MSDOS* Versions •		
• XASMZ80	Z-80				
• XASM85	8085				
XASM05	6805				
XASM09	6809				
XASM18	1802		\$250.00 each		
XASM48	8048/8041				
XASM51	8051	\$200.00			
XASM65	6502	each			
XASM68	6800/01				
XASMZ8	Z8				
XASMF8	F8/3870		\$300.00		
XASM400	XASM400 COP400		\$300.00 each		
XASM75	NEC 7500	\$500.00			
Coming soon: XA	SM68K68000				

(Upgrade kits will be available for new PROM types as they are introduced.)

Programmer	889
Options include:	
Software Driver Package	
· enhanced features, no installation	
 required. 	
• CP/M-80 Version \$	75
• IBM PC Version \$	95
RS 232 Cable	30
8748 family socket adaptor \$	98
8751 family socket adaptor \$1	74
• 8755 family socket adaptor \$1	35

• G7228 Programmer by GTek -- baud

to 2400 ... superfast, adaptive programming algorithms ... programs 2764 in one
minute.

• Programmer \$499

Ask us about Gang and PAL programmers.

• HEXTRAN Universal HEX File Con-

verter -- Converts to and from Intel,
Motorola, MOS Technology, Mostek,

RCA, Fairchild, Tektronix, Texas
Instruments and Binary formats.

• Converter, each version \$250

Call Us

If you're thinking about development systems, call us for some straight talk. If we don't have what you need, we'll help you find out who does. If you like, we'll even talk about Bach,

CALL TOLL FREE 1-800-448-8500

(In the U.S. except Alaska and Hawaii)

VISA and Mastercard accepted. All popular disc formats now available -- please specify. Prices do not include shipping and handling -- call for exact quotes. OEM INQUIRIES INVITED.

*Trademark of Digital Research **Trademark of Microsoft



DEPT. 884-DDJ 804 SOUTH STATE STREET DOVER, DELAWARE 19901 302-734-0151 TELEX 467210

SCISTAR: Greek and Math Symbols with WordStar

e both bought new printers recently and were faced with learning a new set of user-programmed commands that would allow us to make use of some of the special features of these printers. Why not have a special set of commands that is universal—that is, the same for all printers—included with WordStar? Anyone who has used a text formatter such as SuperScript will recognize the value of such a command set. In such systems, a Greek alpha is embedded as \AL\; for Not Equal To, you just write \NE\; and so on. All the special codes for particular printers are transparent and invisible to the operator.

You can set up this same sort of command set very easily with Word-Star, using the option MAILMERGE in a way probably never envisaged by MicroPro (yes, you can do more with WordStar than write hundreds of personalized letters to your relatives). At the same time that you are embedding these codes, you can include some of the special features of your printer, such as emphasized print or underscoring, in a much neater way than Word-Star can. And the automatic wordwrapping still works perfectly, too. Furthermore, the use of universal and meaningful embedded commands greatly enhances the on-screen readability of scientific text.

Any Computer, Any Printer

One of us has a KayPro II with a ProWriter printer, the other a Morrow

by John Thomas and Guy Fletcher

John Thomas, Brookhaven Instruments Corp., 200 13th Avenue, Ronkonkoma, NY 11779.

Guy Fletcher, The School of Mathematics & Physics, Macgarie University, North Ryde, NSW 2113, Australia.

MD2 with a Mannesmann Tally 160L printer. You can use these special commands with any printer. Of course, there is a limited number (too limited!) of user-programmable functions in WordStar, and that number determines how many features you can fit in. First, there are the four user functions, USR1 to USR4, called by ^PQ, ^PW, ^PE, and ^PR. Then there are the ribbon-change toggle (two codes) on 'PY and the character-pitch change on 'PA and 'PN. And that's it: eight available code sequences—and one of those doesn't work from MAIL-MERGE (^PN sends a <CRLF>, for some reason that is obscure to us), though it works fine from WordStar. If each special feature needs one code sequence to turn it on and another to turn it off, then you can run only three special features, but that covers a lot of possibilities in alternate character sets.

The Set of Commands

The set of commands should be universal, so that you can take your disk to another computer and have it read and print your file. And you can sit down at a friend's computer without having to know what has been programmed on which keys or how to get Greek symbols. We tried to think of most of the common needs. Our complete set is listed in Table Ia and Ib (page 26 and 27).

Your printer doesn't offer some of these features? No matter, neither does ours. Include them anyway against the time when you have a better printer, and, meanwhile, print a blank (^PO) to leave space for hand-scripting them in. That way WordStar won't blow up when you print from someone else's disk. To embed any command in your text, just enclose the variable name in ampersands—for example, &al& for alpha. Because upper-case and lower-

	SCIST/ Lower	AR Greek Symb Case	ools Upper C	Case			
Alpha	AL	α	BAL	A			
Beta	BE	β	BBE	В			
Gamma	GA	γ	BGA	Γ			
Delta	DE	δ	BDE	Δ			
Epsilon	EP	ϵ	BEP	E			
Zeta	ZE	B-121-5-1	BZE	Z			
Eta	ET	η	BET	Н			
Theta	TH	θ	BTH	θ			
lota	10	and the state of t	BIO	I			
Kappa	KA	K (1)	BKA	K			
Lambda	LA	λ	BLA	Λ			
Mu	MU	μ (Ε	BMU	M			
Nu	NU	ν 🖟	BNU	N			
Xi	XI	ξ	BXI	Ξ			
Omicron	ОС	0	BOC	0			
Pi	PI	π	BPI	П			
Rho	RH	ρ	BRH	P			
Sigma	SI	σ	BSI	Σ			
Tau	TA	au	BTA	T			
Upsilon	UP	υ	BUP	Υ			
Phi	PH	ϕ	BPH	Φ			
Chi	CH	X	BCH	X			
Psi	PS	ψ	BPS	Ψ			
Omega	OM	ω	вом	Ω			
Table la							

case variables are treated the same, we chose &bal& ("big alpha"), etc., for the upper-case letters.

The Command File SCI.CMD

MAILMERGE requires two dot commands at the head of the document you want to merge-print: a data file instruction, .DF, to specify the file containing the actual codes, and a read variables instruction, .RV, to read the data file. We have about 100 different variables, which makes for several lines of typing, so we put them all in a command file. Insert this at the head of your text with ".FI SCI.CMD". This command file is the same for everyone and is given in Table II (page 27). There is nothing special about the order of the variables, but the data file must have the same order.

The Data File SCI.DAT

The file is individual to your printer. First, you have to decide what features you want and which patch areas to use for each feature.

For the Mannesmann Tally printer, we use ^PQ and ^PW to turn the emphasized (boldface) print option on and off. This is much better than using boldface from WordStar. We use ^PE and ^PR to turn the alternate (Greek) character set on and off, by setting and clearing bit 8 at the printer. We use the ribbon toggle 'PY to turn the in-line underscoring option on and off; again, this works much better than underscoring from WordStar. And we use ^PA and ^PN to disable and enable the paper-out switch, for printing single sheets. We have to use these two controls from WordStar directly, as they are not part of the MAILMERGE file—they wouldn't work from there, anyway!

Likewise, for the ProWriter we prefer to use its own boldfacing capability via ^PQ (on) and ^PW (off). ^PE turns on the Greek and ^PR turns it off (actually, it turns the ASCII character set back on). Again, we use ^PY to toggle the ProWriter's own in-line underscoring on and off. At the moment, we aren't using ^PA and ^PN for anything useful, but we could use them (outside of MAILMERGE) to enable elite characters, for example.

To write the data file, it helps to understand how WordStar processes the hex codes that are stored in your text file or embedded through MAIL-MERGE. All the printable characters from 20H to 7EH are sent directly to the printer. The 26 codes from 01H to 1AH are the 26 control codes ^A through ^Z; WordStar operates directly on these. Thus, 13H (^S, underscore) sends a backspace 08H followed by the underscore character 5FH after every character until you turn off underscoring. ^Q, ^W, ^E, ^R, ^Y, and ^A (but not ^N if you are in MAIL-MERGE) send the code sequences that you have patched in WordStar to the

printer.

The files SCI.CMD and SCI.DAT are most conveniently written under WordStar in nondocument (N) mode. Put a hard <CRLF> at the end of every line; this is especially important to include with the last line in each file. To write ^Q into your data file, you must type ^PQ so that ^Q is actually printed; otherwise, WordStar will act on ^Q directly instead of printing it to the data file.

The two tables in the appendix (page 29) show the actual data files SCI.DAT used for each of our printers.

	S	CISTAR Mat	h Symbols		
Multiply	x	×	Left Arrow		
Divide	DI	1000年	Right Arrow	B	
Plus/Minus	PM	<u>±</u>	Up Arrow	ΰ	1
Not Equal	NE	≠	Down Arrow	D	
Identical	ID	· 沙量 () 多点:	Dagger	DA	†
Approx. Equal	AE	~ ≥	Half	HA	1/2
Greater/Equal	GE		Quarter	QU	1/4
Greater/Order	GO	≥ ≥			
Less/Equal	LE	\leq	Up 0	UO	0
Less/Order	LO	≦ ≲	Up 1	U1	1
Approx.	AP	*	Up 2	U2	2
			Up 3	U3	3
Infinity	IF	Z	Up 4	U4	4
Proportional	PR	x	Up 5	U5	5
Centre Dot	CD		Up 6	U6	6
Convolution	CV	0	Up 7	U7	7
Degrees	DG	0	Up 8	U8	8
Square	SQ		Up 9	U9	9
Grad	GR	∇	Up n	UN	n
Partial Deriv.	PD	∂	Up left brckt	UL	
Square Root	RT	V	Up right brckt	UR)
Therefore	TF		Up Plus	UPL	+
Integral	INT	5	Up Minus	UM	
			Up Dot	UD	•
Bold On	BON		Up Asterisk	UA	*
Bold Off	BOF		Up Slash	US	1
Underline	UND	(toggle)			
		Table I	b		

Command File SCI.CMD

.DF SCI.DAT

•RV AL, BE, GA, DE, EP, ZE, ET, TH, IO, KA, LA, MU

•RV NU, XI, OC, PI, RH, SI, TA, UP, PH, CH, PS, OM

•RV BAL,BBE,BGA,BDE,BEP,BZE,BET,BTH,BIO,BKA,BLA,BMU

•RV BNU,BXI,BOC,BPI,BRH,BSI,BTA,BUP,BPH,BCH,BPS,BOM

•RV BON, BOF, UND

•RV X,DI,PM,NE,ID,AE,GE,GO,LE,LO,AP

•RV IF, PR, CD, CV, DG, SQ, GR, PD, RT, TF, INT

•RV L,R,U,D,DA,HA,QU

•RV U0,U1,U2,U3,U4,U5,U6,U7,U8,U9,Un

•RV UL,UR,UPL,UM,UD,UA,US

Table II

If you have a different printer, you will have to discover your own code sequences to use and, of course, patch them into WordStar. In the appendix we also show how to patch WordStar to configure your printer in the same way as ours.

The Problems

You may have a printer different from either of the ones we've discussed, but you can still learn from our early mistakes with each printer. At the very least, you will become acquainted with the macabre way in which the eccentricities of WordStar and your printer may interact.

Mannesmann Tally 160L

Invoke the D-codes (daisy) option on the printer and patch ROLUP and ROLDOW in WordStar to ESC D and ESC U, respectively (yes, that *is* right). Then you can use the normal superscript and subscript toggles in Word-Star. You *must* install WordStar if you have a teletype-like printer that can backspace; otherwise, you will not be able to overstrike or backspace in any line containing a subscript because of the way the printer handles superscripts and subscripts. You cannot backspace inside a superscript or subscript either; this fact unfortunately prevents superscripting composite characters such as ϕ (overprint o and 1).

Underscoring with &und& is done in-line by the printer—without that repeated backspacing that shakes the whole table—and boldfacing with &bon& and &bof& is done by a smooth extra pass over the line to boldface the required words. Don't try to avoid repeated backspacing by installing WordStar for a nonbackspacing teletype-like printer, because WordStar will then send an overprint line after the printer has rolled down to the subscript line!

When printing a line containing superscripts and subscripts, the Mannesmann Tally 160L printer first rolls down a full line, then prints the superscripted part of the line, rolls down a quarter of a line, prints the main line, rolls down another quarter of a line, and finally prints the subscripted part of the line before rolling down a full space to the next line. So for every line containing superscripts or subscripts, the printer rolls down an extra quarterline: all of this is unknown to Word-Star. This may well take you past the perforation at the end of a page, unless you shorten the page length from 66 to, say, 62 lines. Also, you must use the form-feed option when printing, to position the top of the next page correctly.

There is a fundamental problem of incompatibility between backspacing and superscripting or subscripting in the same line. You just can't superscript around or after a backspace or subscript before or around one. Those few backspaces hidden in SCI.DAT for making composite characters by overstriking are included in this conundrum. The only way a round this seems to be to overprint a complete new line from WordStar (^P RETURN) that contains the desired superscripts and subscripts.

ProWriter

The ProWriter needs to be installed

OF STANDON OF STANDON

Star-Edit and Disk-Edit

Programming Tools as Good as You Are.

You've got the skills. Now, get the tools that match them. Star-Edit and Disk-Edit were designed for the professional programmer; they're powerful, no-nonsense tools that translate your work into fast, effective action.

Star-Edit Text Editor

A powerful, flexible screen editor that includes an outstanding array of text editing commands. With Star-Edit, you can move and reproduce blocks of text or code; view two files simultaneously through split screen windows and move blocks of text or code between different files; perform forward and backward string searches; move forward or backward by character, word, sentence or paragraph; and much more . . . and do it all with just a few keystrokes.

Disk-Edit Disk Utility

A uniquely powerful disk data manipulation tool that gives you access to every bit of information on your disk in both HEX and ASCII. Disk Edit displays byte locations, and side-by-side windows containing disk data in HEX and ASCII. Edit in HEX or ASCII, or switch back and forth between them; edit in either, and changes will be made in both. A valuable utility tool with a full range of text-editing commands. Disk-Edit works on all disk drives, including hard disks.

Both Star-Edit and Disk-Edit are available for CP/M-80, CP/M-86, MS DOS and PC DOS.

For programming tools as good as you are, get Star-Edit and Disk-Edit from...



Available at fine dealers everywhere or directly from SuperSoft. Call 1-800-762-6629. Visa, MasterCharge and American Express accepted.

FIRST IN SOFTWARE TECHNOLOGY
P.O.Box 1628 Champaign, IL 61820 (217) 359-2112 Telex 270365

Circle no. 63 on reader service card.

carefully, though not especially because of our scheme. It has to do mainly with the way in which the ProWriter prints its Greek characters. Basically, you cannot have multiple passes over any whole line of text that has a Greek character in it, because when Word-Star makes multiple passes (e.g., to boldface), it inserts spaces (20H) in any position that is not being overprinted. This normally is no problem, but when Greek is enabled, the Pro-Writer prints ∝ when it receives 20H because WordStar always remembers which character set was enabled at each and every point in the line.

To circumvent these problems, you must install the ProWriter as a teletype-like printer that can backspace, and you must specify that superscripting and subscripting be done via WordStar's ROLUP and ROLDOW. This means that the printer must be initialized (via PSINIT) to enter the incremental print mode, and you need to set up ROLUP and ROLDOW to move a half-line up and down (details in the appendix).

MAILMERGE will not pass " (22H) as a variable, so to get Greek γ from the ProWriter, you have to do it longhand via WordStar (i.e., ^PE"^PR).

By the way, the ProWriter's own boldfacing and underscoring work beautifully in a single line pass, but rolling up and down is a flaky operation on our ProWriter. The printer rolls up erratically; often, the vertical registration can be out by a quarter of a character (big minus for ProWriter).

It Works

We have already written a couple of scientific papers using this scheme. It works beautifully, producing readable source files and the desired output. For example,

x^T&al&^T &ge& &be&&u2& is an &bon&inequality&bof&

produces this line:

 $x^{\alpha} \geq \beta^2$ is an inequality.

Nice, huh?

Reader Ballot Vote for your favorite feature/article Circle Reader Service No. 192.

DD.I

APPENDIX

Data File SCI.DAT for Mannesmann Tally 160L

^E^R,^Ea^R,^O,^Ek^R,^En^R,^O,^O,^Ei^R,i,^O,^O,^Ef^R O, O, O, Ec R, O, Ee R, Eg R, O, o H, O, O, O A,B, Eb R, \ HL,E,Z,H, O,I,K, O,M N, O, O, EoR, P, EdR, T, Y, EhR, X, O, EjRQ, W, Y X, Ev^R , Eq^R , = H/, Ep^R , O, Er^R , $> H^V^V$, Es^R , $< H^V^V$, Ew^R $\hat{E}(R, \hat{O}, \hat{E}^R, \hat{O}, \hat{H} + \hat{E}(R, \hat{O}, \hat{T} - \hat{T}^H, \hat{O}, \hat{H}^E(R, \hat{O}, \hat{E}^T, \hat{T}^H, \hat{V}_u)$ 0, 0, 0, 0, - H, 1/2, 1/4 ^Ex^R,^T1^T,^E)^R,^T3^T,^T4^T,^T5^T,^T6^T,^T7^T,^T8^T,^T9^T,^E^R ^T(^T,^T)^T,^T+^T,^T-^T,^T.^T,^T*^T,^T/^T

Wordstar Patches for Mannesmann Tally 160L

Bold_on	USR1	ESC[=z	04H,1BH,5BH,3DH,7AH
Bold_off	USR2	ESC[>z	04H,1BH,5BH,3EH,7AH
Greek_on	USR3	ESC[9z	04H,1BH,5BH,39H,7AH
Greek_off	USR4	ESC[8z	04H,1BH,5BH,38H,7AH
Paper_out_disable	PALT	ESC[:z	04H,1BH,5BH,3AH,7AH
Paper_out_enable	PSTD	ESC[<z< td=""><td>04H,1BH,5BH,3CH,7AH</td></z<>	04H,1BH,5BH,3CH,7AH
Underscore_on	RIBBON	ESC[4m	04H, 1BH, 5BH, 34H, 6DH
Underscore_off	RIBOFF	ESC[0m	04H,1BH,5BH,30H,6DH
Superscripting	ROLUP	ESCD	02H,1BH,44H
& subscripting	ROLDOW	ESCU	02H,1BH,55H
	11020011	LUCU	02H, IDH,35H

Printer installed as "teletype-like printer that can backspace" D-codes enabled on printer

Data File SCI.DAT for Prowriter

^E^R,^E!^R,^O,^E#^R,^E\$^R,^E%^R,^E&^R,^E'^R,^E(^R,^E)^R,^E*^R,^E+^R ^E'',''^R,^E—^R,^E.^R,^E/^R,EO^R,^E1^R,^E2^R,^E3^R,^E4^R,^E5^R,^E6^R,^E7^R A,B, E9 R, E8 R,E,Z,H, O,I,K, E; R,M N, O,O, O,P, E: R,T,Y, O,X, P, E< R Q, W, Y X, \hat{O} , $\hat{E}\hat{D}$, \hat{R} , $\hat{E}\hat{E}$, \hat{O} , \hat{O} , $\hat{E}\hat{F}$, \hat{C} , $\hat{C$ `EK^R,^O,^E1^R,^EJ^R,^EO^R,^E?^R,V^H^T—^T,o^H,`E>^R,^EL^R,^O EBÎR, ECÎR, E@ÎR, EAÎR, ÎH-, ÊMÎR, ÊNÎR `EO^R,`EP^R,`EQ^R,`ER^R,`ES^R,`ET^R,`EU^R,`EV^R,`EW^R,`EX^R,`Tn^T EY^R, EZ^R, E|^R, E\^R, E|^R, E^R, E_^R

WORDSTAR PATCHES for ProWriter

Bold_on	USR1	ESC!	02H,1BH,21H
Bold_off	USR2	ESC"	02H,1BH,22H
Greek_on	USR3	ESC&	02H,1BH,26H
ASCII on	USR4	ESC\$	02H,1BH,24H
	PALT PSTD		
Underscore_on	RIBBON	ESCX	02H,1BH,58H
Underscore off	RIBOFF	ESCY	02H,1BH,59H
Superscripting	ROLUP	ESCr <lf></lf>	04H,1BH,72H,0AH,00H
& subscripting	ROLDOW	ESCf <lf></lf>	04H,1BH,66H,0AH,00H
Normal < CRLF>	PSCRLF	ESCf <cr><lf><lf></lf></lf></cr>	05H,1BH,66H
			ODH,OAH,OAH
Initialization	PSINIT	ESC[ESCT12	07H,0DH,1BH,5BH
string		建 的基件。据为2000年代	1BH,54H,31H,32H
Finish string	PSFINI	ESC]	02H,1BH,5DH

Printer installed as "teletype-like printer that can backspace" PSINIT sets ProWriter in incremental mode and sets linefeed to 1/12".

PSFINI returns ProWriter to logic seek print mode.

PSCRLF gives a "normal" < CRLF> by making sure that the carriage rolls down, and sending two 1/12" linefeeds.

What's the Diff?

A File Comparator for CP/M Plus

file comparator is a program that compares two files and reports the differences between them, or reports that they have no differences. Such a program can be very useful, especially in systems that don't put a time stamp on each file that is created. In such systems a comparator is often the only way to answer such questions as, "Is that *really* the latest version?".

A file difference finder is a program that goes one step further. Not only does it find the differences between two files, it records those differences in a third file for later use. A difference finder is a key part of any system for the automatic management of software development. A program source file goes through many changes as it is developed, tested, and maintained. With a difference finder, each successive version of a program can be stored as a difference file that contains only the changes from the prior version. There's great storage economy in this. Furthermore, a difference file is a compact representation of a program update, and as such it can be a good way of distributing a bug-fix.

The ideal format for a difference file is a script of commands that will direct some kind of editor in the process of converting the old file into the new one. When differences are captured in such a form, the latest version of a base file can be created entirely by software, automatically. The editor is applied to the base file and given successive difference files as its script. The edited result is the new file.

For example, in IBM's VM/370 operating system difference files are called "update" files, and there is a

by D.E. Cortesi

D.E. Cortesi, 430 Sherman #212, Palo Alto, CA 94306.

special UPDATE program that will generate a new file from a base file and a list of update files. Text editors used with the system have the ability to store only the changes made during an edit session, writing them as an update file.

A system like the one in VM/370 requires special-purpose software (the UPDATE program is used only for that, and update files are not the normal output of an edit). A public domain program called DIFF can be found on some bulletin boards; it, too, seems to use a special-purpose program to apply the updates. How much nicer it would be if a file difference system could be constructed using only the standard tools a system affords! No doubt this can be done in Unix, with its plethora of tool programs and its great ease of plugging tools together. But it can also be done in CP/M Plus.

CP/M Plus contains most of the tools that are needed. It has ED, an editor that nobody would willingly adopt for personal use but one that is perfectly suitable as an automatically driven file updater. Furthermore, CP/M Plus contains the command GET, which redirects console input to a file. Thus these two CP/M Plus commands:

get differ.dat ed basefile

would cause ED to load BASEFILE and perform upon it the modifications commanded by the lines of DIFFER DAT. If there were several difference files to be applied in order, they could be handled by a sequence of commands, and the sequence could be written up as a submit file to run automatically. All that is lacking is an automatic way to create difference files.

What is needed is a program that will read an old and a new file (most likely the new file will be the result of editing the old one, and the old file will be the .BAK version of the new one). After comparing the two files, the program will write a difference file; this will contain exactly the ED command lines needed to transform the old file into the new one.

It sounds fairly straightforward, doesn't it? Before you read any further, take a few minutes to sketch out the top-level logic of such a program. I'll wait.

Ah, you're back. Not so simple after all, is it? As a matter of fact, it's a bear. The intuitive algorithms (and some that are not so intuitive) all seem to lead to excessive storage use, or to execution times that are an exponential function of the file sizes. One common method uses a comparison "window" whose size is a critical program parameter. Here's a quote from the manual for AUTODIFF, a file comparator distributed by The Software Toolworks.

"AUTODIFF starts to work by lining up the tokens from one input file ... with the tokens from the other input file. Next it looks for the start of a difference The next task is the hard part: finding where the difference ends. This is called synchronizing the files, and AUTODIFF's tool for accomplishing it is the sync window Starting at the head of each queue, AUTODIFF slides a sync window back and forth over each queue until it finds the same pattern showing through both windows, or until one or both of the windows slide off the end of its queue.

"If the sync windows were very small... AUTODIFF would most likely see the same pattern in both windows even though the surrounding tokens were totally different. On the other hand, if the sync windows were very large... AUTODIFF would most likely slide both windows right off the ends of their queues, in which case its idea of a difference would consist of the whole of each input file. So the size

of the sync window is very important. For AUTODIFF to work, it can't be too large or too small.... Unfortunately, the right sync window size depends heavily on the statistics of the data...." (Copyright 1982, Digital Constructs, Inc.)

After making a number of stabs at the problem, I decided to turn to "the literature" to find out if somebody smarter than I had found a better answer. And somebody had. The somebody is one Paul Heckel, who published his algorithm in a paper entitled "A Technique for Isolating Differences Between Files" in the April 1978 issue of Communications of the ACM. Heckel's algorithm actually generates more data than a difference finder needs. Not only that, but it does it in time that is guaranteed to be linear in the size of the files and with storage that is never more (usually much less) than the combined file sizes.

Heckel started by reversing the terms of the problem. Rather than searching for differences between two files, his algorithm concentrates on finding the similarities. Once those are marked, whatever is left is a difference. He devised two rules that would find most similarities.

Rule 1 says that "a line that appears once and only once in each file must be the same line (unchanged but possibly moved)." The rule is implemented using a symbol table and two arrays.

First, the old file is read and each of its lines is entered into the symbol table. An array called *OA*, indexed by the line numbers of the file, is used to record the symbol-table numbers of each line. Only the text of unique lines needs to be retained; duplicated lines simply get duplicate symbol-table indices. A counter in each symbol-table entry is incremented each time a line is found.

Next the new file is read and its lines are entered to the symbol table. Only line texts unique to the new file need to be retained. Another counter in each symbol-table entry is incremented to record the appearance of each new-file line, and the file is recorded as an array *NA* of symbol-table indices.

Now rule 1 can be applied. For each line represented by NA, look at the corresponding symbol-table entry. If the old and new counters both contain

1, that line satisfies rule 1 (appears once and only once in each file). The corresponding new and old lines are marked as "matched"; they cannot be differences. Lines are "matched" by putting reciprocal line numbers in their NA and OA array entries.

At this point, four kinds of lines are known. Those that satisfy rule 1 are marked. The remainder either appear in the old file but not the new (and hence are deletions); or appear in the new file but not the old (these are insertions); or appear more than once in one or both files (and are ambiguous at this point). To understand this, think of the block comments in a program file. The comment text lines, if they are unchanged, would be matched up by rule 1. The decorative lines of comment delimiters that frame them would not be matched by rule 1 because many of them appear in each file.

Heckel's rule 2 acts to resolve the last category. I would phrase it this way: "If the lines adjacent to a matched pair of lines are identical, then the adjacent pair are matching lines as well." These adjacent matches can be found in two sweeps over the arrays, one to find matches that follow a matched line and the other to find those that precede one. Afterward, all the similar lines have probably been found. The only similarities that would be missed are those that (a) appear more than once in each file and (b) are separated from every rule-1 line by a changed line. Such missed similarities will not be lost, but they will be treated as changed lines in the output.

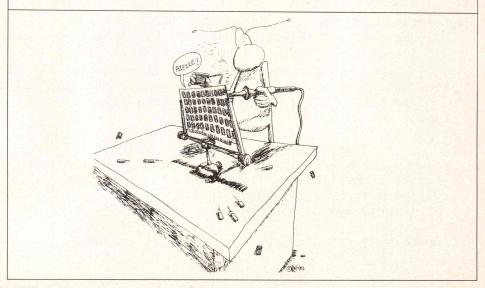
Heckel claims as an advantage of his algorithm that it recognizes block moves for what they are. Lines that are moved, but not otherwise changed, are matched as similar. They can be recognized by discontinuities in the ascending sequence of line numbers in both the OA and NA arrays. Unfortunately, there is no easy way to record a block move as a sequence of ED commands, so I couldn't use the information. However, it is possible to locate all block moves and "unmatch" them (converting them into deletes and inserts) in a single sweep over the arrays.

The listing that follows (page 32) is a prototype implementation of Heckel's algorithm in Pascal. Since it was put together as a learning exercise, it has a number of rough edges; still, it does work. Except for a few compiler dependencies (declaring a variable-length string, finding its length, and opening files) it should work with any CP/M or MSDOS Pascal compiler.

Pascal is not the best language for this program. It really needs more freedom to allocate storage. An implementation in C could be more flexible about maximum file sizes. An assembly language version could play several tricks to reduce the size of the symboltable entries as well. Even so, this version can do useful work on files of moderate size, and it is surprisingly quick in execution, running in only a little more time than it takes to read the two files and write the difference file.

(Listings begin on next page)

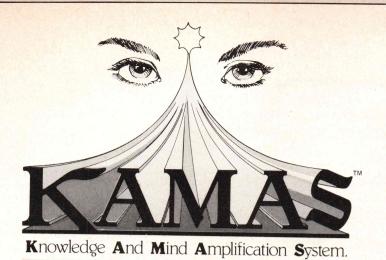
Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service No. 193.



What's the Diff? Listing (Text begins on page 30)

```
HDIFF: a file-difference finder
   Based on "A Technique for Isolating Differences
   Between Files" by Paul Heckel, Communications of
   the ACM Vol 21 Number 4 (April 1978), this program
   (1) reads an old and a new file, (2) finds the diff-
   erences between them in time that is linear in the
   size of the files, and (3) writes a file which is
   a script to drive an editor (ED) to change the old
   into the new.
program diff(input,output);
label 99; { main-proc error exit }
const
   maxfile = 300; { largest old/new file (arbitrary) }
   maxover = maxfile+1;{ allow for a sentinel value }
   maxchar = 127; { max chars/line, 128 bytes/string }
                        { symbol table size, prime > 2*maxfile }
   maxsym = 1023;
    topsym = maxsym-1; { max index of ST array }
{ CP/M storage requirements are: 6*maxover + 8*maxsym plus
  128 bytes per each UNIQUE line in the combined files }
type
    symnum = 0..topsym; { type of symbol table indices }
    linenum = 1..maxover; { types of indices to OA, NA }
    linecht = 0..maxover:
{ compiler dependent: define a varying string up to maxchar bytes }
    ltext = string maxchar;
                       { descriptor of one file line in OA, NA }
    linerec = record
        matched: boolean; { true when matched with other file }
        index: integer; { index to symtab or to other file }
    end;
    symrec = record { one symbol table entry }
        hashval: integer; { hash of this line, neg = unused entry }
                            { address of text string }
        lineval : ^ltext;
        Oline : linenum; { index to line in old file OA }
        Ocount: 0..2; { occurrences in old file: 0, 1, many }
Nount: 0..2; { ditto in new file }
    end;
var
    oldmax, newmax : linecnt; { linecount of each file }
    oldfile, newfile, diffile: text;
    OA, NA : array[linenum] of linerec;
    ST : array[symnum] of symrec;
{ Compute a 15-bit signature based on all the characters of a
```

(Continued on page 34)



☐ Get a head start at developing applications in the exciting, new area of **Outline Processing.** With KAMAS™, you can organize ideas in a familiar, outline form. And retrieve them with astonishing speed using the built-in KAM™ Access Method.

 \square All under the precise control of an extensible, programming environment. The KAMAS $^{\infty}$ language produces compact, threaded code and is integrated with the Outline Processing. Source code can be entered with the built-in text editor and stored in outline form, providing extraordinary leverage for structured programming and development.

☐ The language is highly interactive and fast, offering an outstanding environment for developing and testing applications using the Outline Processing, Information Retrieval, Word Processing, and Telecommunications features.

☐ Capitalize on the next wave of the software revolution which promises to surge as high as Spreadsheet Processing. Available for Kaypro computers. Special introductory offer: \$147. Send now for your **free** copy of "The KAMAS Report."



COMPUSOPHIC SYSTEMS

Dept. 121 • 2525 SW 224th Ave Aloha, Oregon 97006 • [503] 649-3765

KAMAS is a trademark of Compusophic Systems. Kaypro is a trademark of Kaypro Corporation.

Circle no. 11 on reader service card.

Introducing the Creative Genius...

Discover how easy programming can be with DataBurst:

A unique runtime screen processor and source program generator, DataBurst™ will decrease your program development time and increase the value of your application programs. The unique DataBurst™ screen editor provides fast, easy screen design. Program independent screen formats reduce both development and maintenance time.

During execution of your program, DataBurst™ controls all user interaction through one assembly language interrupt service routine, requiring as little as 14K of memory. A true full-screen processor, DataBurst™ allows unlimited design complexity, and brings a mainframe advantage to your IBM® PC.

DataBurst** is available through your local computer retailer or directly from Key Solutions, Inc. To order directly, please send check or money order for \$225* to Key Solutions, Inc., P.O. Box 2297, Santa Clara, CA 95055. Additional language support (BASIC Compiler and C Compiler) is available for \$40* (Please inquire about release dates for other language interfaces).

IBM[®] is a registered trademark of IBM Corporation. DataBurst[™] is a trademark of Key Solutions, Inc.

Copyright 1984 Key Solutions, Inc.

The Design Tool for the Creative Programmer



FOR THE IBM PERSONAL COMPUTER.

THE PREMIER LANGUAGE OF ARTIFICIAL INTELLIGENCE FOR YOUR IBM PC.

DATA TYPES

Lists and Symbols Unlimited Precision Integers Floating Point Numbers Character Strings Multidimensional Arrays Files Machine Language Code

MEMORY MANAGEMENT

Full Memory Space Supported Dynamic Allocation Compacting Garbage Collector

FUNCTION TYPES

EXPR/FEXPR/MACRO Machine Language Primitives Over 190 Primitive Functions

IO SUPPORT

Multiple Display Windows Cursor Control All Function Keys Supported Read and Splice Macros Disk Files

■ POWERFUL ERROR RECOVERY

- 8087 SUPPORT
- **COLOR GRAPHICS**

LISP LIBRARY

Structured Programming Macros Editor and Formatter Package Support Debugging Functions .0BJ File Loader

RUNS UNDER PC-DOS 1.1 or 2.0

IQLISP

51/4" Diskette and Manual _____ \$175.00 Manual Only ____ \$ 30.00

∫q Integral Quality

P.O. Box 31970 Seattle, Washington 98103-0070 (206) 527-2918

Washington State residents add sales tax. VISA and MASTERCARD accepted. Shipping included for prepaid orders.

```
line. Pick a symbol table entry based on the hash and return
 the symbol index. If different lines hash to the same entry
  just overlow by ones, the simplest policy. The full hash
 code is saved in the symtab entry for quick discovery of
 unequal lines. }
function store(var t: ltext) : symnum;
    label 1,2,3;
    var s: symnum; h: integer;
    { compiler-dependent: return length of string t }
    function strlen(var t: ltext): integer;
        type pascalz = string 255;
        function length(pz: pascalz):integer; external;
        begin strlen := length(t) end;
    function hash(var t: ltext): integer;
        const hashlim = 16256; { (maxint div 4)-127 }
        var q: integer; len: 0..maxchar;
        begin
            q := 0;
            for len := strlen(t) downto 1 do begin
                q := q + ord(t[len]); { add current byte }
                { shift left to 14 bits, then wrap }
                if (q < hashlim) then q := q + q
                                 else q := 1 + q \text{ div } 2
            end;
            hash := q
        end; { function hash }
    begin { function store }
        h := hash(t);
        s := h mod maxsym; { initial probe of symbol table }
     { find duplicate line or vacant symbol starting at ST[s].
      If you know a "structured" way to code this, let me know. }
     1: if (ST[s].hashval < 0 ) then goto 2; { free entry }
         { used entry, is t a duplicate of another line? }
        if (ST[s].hashval = h) { quick test, eliminates most }
         and(ST[s].lineval = t) { slow test assures dup line }
             then goto 3; {yes, we have this line}
         s := (s+1) mod maxsym; { try next table entry }
         goto 1;
     2: { empty symbol table entry found -- install new line }
         with ST[s] do begin
             hashval := h;
             new(lineval);
             lineval := t
         end:
```

```
3: { line now exists in symbol table }
    store := s
    end:
{ read the old file, store its lines in the symbol table, and
  count their occurrences in Ocount in each entry. }
procedure pass1;
    var o: linecnt;
        s: symnum;
        t: ltext;
    begin
        0 := 0;
        repeat
            readln(oldfile,t);
            0 := 0+1;
            s := store(t);
            with ST[s] do begin
                Oline := 0;
                if (Ocount <> 2) then Ocount := Ocount + 1
            end;
            with OA[o] do begin
                matched := false;
                index := s
        until ( eof(oldfile) or ( o = maxfile ) );
        with OA[o+1] do begin { create sentinel }
            matched := true:
            index := maxover
        end;
        oldmax := o
    end;
{ read the new file, store its lines in the symbol table, and
 count their occurrences in Ncount in each entry. }
procedure pass2:
   var n: linecnt;
        s: symnum;
        t: ltext;
   begin
        n := 0;
        repeat
            readln(newfile,t);
            n := n+1;
            s := store(t):
            with ST[s] do
                if (Ncount <> 2) then Ncount := Ncount + 1;
           with NA[n] do begin
            matched := false:
           index := s
   until ( eof(newfile) or ( n = maxfile ) );
   with NA[n+1] do begin { create sentinel }
       matched := true;
       index := maxover
```

(Continued on next page)

What's the Diff? Listing (Listing continued, text begins on page 30)

```
end;
        newmax := n
    end;
{ record the fact that two lines match by placing
  reciprocal indices in their array entries. }
procedure matchup(o, n: linenum);
    begin
        with OA[o] do begin
            matched := true;
            index := n
        end:
        with NA[n] do begin
            matched := true;
            index := o
        end
    end:
```

{ apply rule 1: when a line appears just once in each file, it is

Q-PRO 4 blows spread the word

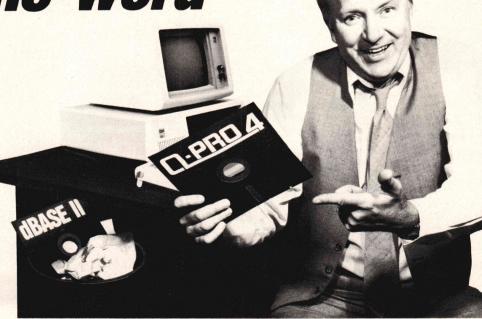
Have you heard?

During the last 12 months, thousands of applications programmers dumped dBASE II.

Why?

Because dBASE II hasn't improved a lick in years. And that makes it a whole generation behind Q-PRO 4... the 4th generation applications development language for microcomputers.

With dBASE II, all the original bugs, complicated operations and absurd restrictions (like only two open files) are still there. dBASE II just can't make it for applications in 1984.



Circle no. 52 on reader service card.

```
the same line (altho perhaps not in its original position).}
procedure pass3;
    var s: symnum; o,n : linenum;
    begin
        for n := 1 to newmax do begin
            s := NA[n].index;
            with ST[s] do
                if (Ocount=1) and (Ncount=1) then
                    matchup(Oline.n)
        end
    end;
{ apply rule 2, sweeping down the file to spread the rule-1 matches
  toward the end of the file. }
procedure pass4a;
    var o, o1, n, n1 : linenum;
    begin
        for n := 1 to newmax-1 do
            if (NA[n].matched) then begin
                n1 := n + 1;
                if (not NA[n1].matched) then begin
                    o := NA[n].index;
```

(Continued on next page)

dbase II away

Apparently, Ashton Tate (the dBASE II merchant) is gambling you don't know any better. It's pitiful.

Well, we've been blowing the whistle on Ashton...and Tate, too.
And you can spread the word.

Be one of the growing legions that's moving up to Q-PRO 4... the complete 4th generation applications development system for microcomputers.

You can use Q-PRO 4's super efficient syntax to finish business programs much faster. And the extensive error trap and help screen capabilities make the finished software products far more friendly, too.

As one convert put it, "Q-PRO 4 has it all...the formatted data entry field edits, and report generator are absolutely superb.

"Any applications programmer still struggling with outdated 3rd generation data base managers or worse, a 2nd generation language like BASIC is ripping himself off."

So what are you waiting for? Here is your chance to dump all the dBASE II hassles and move up to Q-PRO 4... the sensational 4th generation language for faster, easier application development.

You owe it to yourself, your career, and your family to move up to Q-PRO 4 now. It's that good.

Attention Q-PRO 4 Hotshots.
Current version 3.0 includes Multikey ISAM (true mainframe power).

Runs with PCDOS, MS-DOS, CP/M, MP/M, CP/M86, MP/M86, TurboDOS, MmmOST, MUSE, and NSTAR.

Single-user_\$595; Multi-user_\$795. Author's lock up package available. Finished applications are freely transportable between operating systems. Multi-user with true record and file lock.

For Q-PRO 4 demonstration, go to the nearest MicroAge store or other fine dealer.

136 Granite Hill Court Langhorne, PA 19047 (215) 968-5966 Telex 291-765

CP.M. MP.M. CP.M86, and MP.M86 are trademarks of Digital Research, TurboDOS, MmmOST, MUSE, NSTAR, MS-DOS and PCDOS are trademarks of Software 2000, TeleVideo Systems, O.S.M., Molecular, Microsoft and IBM, respectively.

<u>quic·n·easi</u> products inc.

```
if (o < oldmax) then begin
                        01 := 0 + 1;
                        { the "matched" flags participate in
                          the following comparison to ensure
                          that symbol indices are compared }
                        if (OA[o1] = NA[n1]) then
                            matchup(o1,n1)
                    end
                end
           end
   end:
{ apply rule 2, sweeping up the file to spread previous matches
  toward the top of the file. }
procedure pass4b;
   var o, o1, n, n1 : linenum;
    begin
        for n := newmax downto 2 do
            if (NA[n].matched) then begin
                n1 := n - 1;
                if (not NA[n1].matched) then begin
                    o := NA[n].index;
                    if (o > 1) then begin
                        01 := 0 - 1;
                        if (OA[o1] = NA[n1]) then
                            matchup(01,n1)
                    end
                end
            end
    end;
{ Unmatched lines in OA represent deletes; unmatched NA lines are
  inserts. Ignoring these, the matched lines should increase
  monotonically. When they don't, when a discontinuity appears,
  a block-move is present. It is unclear how to record such
  moves in a difference file, so here we find them and "unmatch"
  the moved block, converting it into a delete/insert.
  There is a figure/ground ambiguity -- any block move can be
  seen as a move-up of some lines or a move-down of others. We
  select the smaller of the two groups to be the "moved" group.}
procedure pass5;
    var o, n: linenum;
    procedure resolve (var o, n : linenum);
     { a discontinuity starts at OA[o] and NA[n]. figure out whether
      fewer lines have been moved up to NA[n] or down from OA[o],
      and convert the smaller group to inserts and deletes. }
        var xo, xn : linenum;
             t, ospan, nspan : integer;
             s : symnum;
```

```
xo := o; { measure the block starting at OA[o] }
        repeat
            t := 1 + OA[xo].index;
            xo := xo + 1
        until (t <> OA[xo].index):
        ospan := xo - o;
        xn := n; { measure the block moved up to NA[n] }
        repeat
            t := 1 + NA[xn].index;
            xn := xn + 1
        until (t <> NA[xn].index);
        nspan := xn - n;
        if (ospan < nspan) then begin { a block moved down }
            xo := o;
                                { convert OA[o..] which matches }
            xn := OA[o].index; { lines further down in NA }
            t := ospan; { number of lines to convert }
o := o + ospan { continue scan after block }
        end
        else begin
                                         { a block moved up }
            xn := n;
                                { convert NA[n..] which matches }
            xo := NA[n].index; { lines further down in OA }
            t := nspan; { number of lines to convert }
            n := n + nspan { restart scan after block }
        end;
        { Unmatch a block of matched lines. We need the
          symbol table index here, and find it by scanning
          the table. This expensive operation could be
          eliminated by retaining the symbol index in OA}
        while (t > 0) do begin
            s := 0;
            while (ST[s].Oline <> xo) do s := s + 1;
            with OA[xo] do begin
               matched := false;
                index := s
            end;
            with NA[xn] do begin
               matched := false:
               index := s
            end:
            xo := xo + 1;
            xn := xn + 1;
            t := t - 1
       end
   end; { procedure resolve }
begin { procedure pass5 }
   o := 1; n := 1;
   while (OA[o].index <> maxover) do begin
       while (not OA[o].matched) do {skip deletes}
            0 := 0 + 1;
       while (not NA[n].matched) do {skip inserts}
           n := n + 1;
```

(Continued on next page)

What's the Diff? Listing (Listing continued, text begins on page 30)

```
if (OA[o].index = n) then begin { matching lines }
                0 := 0 + 1;
                n := n + 1
           end
           else { a discontinuity ? }
                if (OA[o].index <> maxover) then { yes }
                    resolve(o,n)
                else { no, just the sentinel }
       end
   end: { pass5 }
{ Write the difference file as a script that will drive a
 line-editor in converting the old file to the new one.
 Actual output is modularized for ease in converting to
 a different editor than CP/M's ED. }
procedure pass6;
   var o, n, xn : linenum;
        t : integer;
    { cause necessary initialization of editor }
    procedure putfirst;
        begin
            writeln(diffile, '#a');
        end;
    { delete t lines starting at current one, leave
     line after last delete as current line }
    procedure putdel(t : integer);
        begin
            writeln(diffile,t:1, 'k')
        end;
    { set up to insert lines at end of file }
    procedure putbot;
        begin
            writeln(diffile, '-b')
        end;
    { skip ahead t lines in the file }
    procedure putmov(t : integer);
        begin
             writeln(diffile,t:1, 'L')
         end;
     { insert t lines beginning with NA[k] -- the new
       lines PRECEDE the current line and afterward the
       current line is the same as at the start. }
     procedure putins(k : linenum; t : integer);
         begin
             while (t > 0) do begin
                 writeln(diffile, 'i', ST[NA[k].index].lineval^);
                 k := k + 1;
```

A>DBPACK: Information Manager -- Great for Mailing Lists, Form letters, Tabulation and organizing data. Supports query, sort/search on multiple keys, report generation and many other data base functions. \$115/\$25.

A>COMCOM: Communication program. Uploads/Downloads files, and more. \$95/\$15.

A>CPMCPM: Transfers files (any type) between CP/M computers with incompatible disks. \$65/\$10 includes copy for each computer.

A>FILER: Archives, Sorts and Catalogs files with substantial disk space savings. \$49.

A>BASXREF: Alphabetizes and Cross-references variables vs. line numbers in BASIC programs. Simplifies program maintenance. \$39.

A>UNERA: Recovers erased files. \$29.

CP/M is a registered trademark of Digital Research, Inc.

- Available in most disk formats.
- Clearly written and indexed manuals included. Where two prices are quoted, second refers to manual only (creditable towards software).
- All packages returnable in 15 days.

COMPU-DRAW 1227 Goler House (716) - 454 - 3188

23208

MasterCard, Visa Amex cards, PO's from Rochester, NY 14620 recognized institutions and COD are welcome.

F 47.4

è

Circle no. 10 on reader service card.

FOR \$29.95, DISK INSPECTOR MAKES YOU A SHERLOCK HOLMES OF THE COMPUTER!

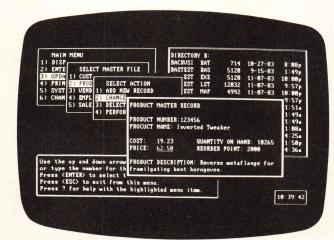
Ever wonder what happened to that erased or lost file? Did text suddenly disappear and can't be found? Did a bad sector do strange things to your files? Then track them down with Disk Inspector! Rated "Excellent" by INFOWORLD, Disk Inspector is a utility that pays for itself with the first recovery. Even more, Disk Inspector allows you to use the Auto-Load feature of CP/M, blank out bad sectors, create multiple entries, small files, all without any knowledge of programming! Just \$29.95, plus \$2.00 postage and handling, you become the chief inspector. Sold with the usual Overbeek 30 day money-back guarantee

MAKE ME A CHIEF INSPECTOR! Enclosed find check for \$31.95 or ____Expires: ___ charge my Mastercard # ____ _Expires: _ Check format desired: _8" SSSD __KayPro II Osborne Single Density __Superbrain _Osborne Double Density _NEC 5" _Northstar Advantage _Morrow Micro Decision _Televideo802 __Northstar Horizon __Xerox 820 Single Density _ALTOS 5 _Apple/Softcard _Xerox 820 Double Density __Epson I'm not ready to order now, but send me information about all the affordable programs from Overbeek Enterprises. Address _____ City ___ _ State _____ Zip ____ OVERBEEK ENTERPRISES, P.O. Box 726D, Elgin, IL 60120

312-697-8420 CP/M is a trademark of Digtital Research. Disk Inspector - another affordable program from Overbeek Enterprises.

Circle no. 42 on reader service card.

WE DO WINDOW



AMBER SYSTEMS, INC. 1171 S. Sunnyvale-Saratoga Road San Jose CA 95129

(408) 996-1883

Copyright 1984 Amber Systems, Inc.

Don't just put your applications in windows-put windows in your applications with VSI-the window manager.

VSI is a high-speed screen management tool. You can create up to 255 simultaneously active overlapping windows-large or small-for any application program. Read to or write from any window and display them with borders and user declared priorities. VSI is callable from any compiled language and supports all color and monochrome video attributes.

Cut Development Time

VSI's powerful primitives simplify your screen management chores with a complete library of functions. And you can preview and edit your screen layout before you actually program it.

But that's only the beginning.

Free Demo Disk

Our free hands-on demo disk will have you doing windows, too. Return the coupon with \$4.50 for postage and handling. MasterCard or Visa accepted with phone orders only VSI is used with IBM PC, XT and compatibles as well as TI Professional, and Wang PC.

I'm enclosing \$4.50 fe	8086/8088 based machines a or postage and handling. Pleas d is attached. (Offer expires De	se send me your free demo
Computer:		
Name		
Company		
Address		
City	State	Zip

```
t := t - 1
           end:
       end:
   { close file and end editor }
   procedure putlast:
       begin
           writeln(diffile, 'e');
       end:
   begin {pass6}
       putfirst;
       o := 1; n := 1;
       repeat
           if (not OA[o].matched) then begin { deleting }
                t := 0;
                repeat
                    t := t + 1;
                    0 := 0 + 1
                until (OA[o].matched);
                putdel(t)
            end:
            if (not NA[n].matched) then begin { inserting }
              { if (n = 1) then puttop? -- nothing special
                needed to insert above line 1 with ED }
                if (o > oldmax) then { ins. at bottom }
                    putbot;
                t := 0;
                xn := n;
                repeat
                    t := t + 1;
                    n := n + 1
                until (NA[n].matched);
                putins(xn,t)
            end;
            t := 0:
            while (OA[o].matched) and
                  (NA[n].matched) and
                  ( o <= oldmax ) do begin { skipping unchanged lines }
                t := t + 1;
                0 := 0 + 1;
                n := n + 1
            end;
            if (t>0) then putmov(t)
        until (OA[o].index=maxover) and (NA[n].index=maxover);
        putlast:
    end;
procedure setup;
```

```
var j : symnum;
        fspec : string 14;
   begin
        for j := 0 to topsym do
            with ST[j] do begin
                hashval := -1; { entry not in use }
                Oline := maxover:
                Ocount := 0:
                Ncount := 0
            end:
       oldmax := 0:
       newmax := 0:
{ compiler-dependent: get names and open files }
       write('ORIGINAL file''s spec: '); readln(fspec);
       reset(fspec, oldfile);
       write('MODIFIED file''s spec: '); readln(fspec);
       reset(fspec, newfile);
       write('DIFF file''s spec: '); readln(fspec):
       rewrite(fspec, diffile)
begin {main procedure: set up, check for empty files, invoke passes }
    setup:
    if (eof(oldfile)) then begin
        writeln('Old file is empty; diff is all-insert.');
        goto 99
    end:
    if (eof(newfile)) then begin
        writeln('New file is empty; diff is all-delete.');
        goto 99
    end;
    pass1; { read, store old file }
    pass2; { read, store new file }
    pass3; { apply rule 1 }
    pass4a; { apply rule 2 working down }
    pass4b; { apply rule 2 working up }
    pass5; { convert moves to del/ins pairs }
   pass6; { generate script for ED }
99:
end.
```

End Listing



An Infinite Key Encryption System

f you have ever stored private information on a public disk file or transmitted it over the telephone network to storage on a mainframe or to a correspondent, you may have wondered if your private information would remain private. Who else is reading it? Your co-workers? Other subscribers? The system programmers? The government?

Trusting souls who have never felt the slightest concern about such matters may read no further; others may be interested to learn how to protect themselves against such intrusion and theft. This article discusses the construction of a strong file encryption system for CP/M, which we call "TNT."

Several encryption programs for microcomputers are now on the market. To our knowledge, however, none of their authors has made public the method used or has offered any argument to support the claim that the system is indeed secure. By its very nature, an encryption system is difficult to test. In practice, "security" only means that the system has been subjected to careful analysis and repeated attack and has remained unbroken.

We describe our system in detail sufficient to allow its implementation on most microcomputers. We also make our design choices explicit and offer argument to support them, so that readers can form their own opinions as to the strength of the system. We make no claim of theoretical breakthrough here, only the careful application of cryptographic principles. Finally, we offer the design and program code to the public, free of any claim of copyright or proprietary right. If our system can withstand critical analysis, then the public will have an encryption system it can trust.

Some Cryptographic Basics

A few definitions are appropriate first. We use the term "encryption" to refer to the general process of making plain information secret and making secret information plain. To "encipher" a file is to transform the information in the file so that it is no longer directly intelligible. The file is then said to be in "ciphertext." To "decipher" a file is to transform it so that it is directly intelligible; that is, to recover the "plaintext."

The two general devices of encryption are "ciphers" and

by John A. Thomas and Joan Thersites

John A. Thomas, 621 N.W. Sixth Street, Grand Prairie, TX 75050.

Copyright © 1984 John A. Thomas and Joan Thersites. All rights reserved.

"codes." A cipher works on the individual letters of an alphabet, while a code operates on some higher semantic level, such as whole words or phrases. Cipher systems may work by transposition (shuffling the characters in a message into some new order), by substitution (exchanging each character in the message for a different character according to some rule), or by a combination of both. In modern usage, transposition is often called "permutation."

Shannon referred to substitution as "confusion" because the output is a nonlinear function of the input, thus creating confusion as to the set of input characters. He referred to transposition as "diffusion" because it spreads the dependence of the output from a small number of input positions to a larger number.

A cipher that employs both transposition and substitution is called a "product" cipher. In general, product ciphers are stronger than those using transposition or substitution alone.

Every encryption system has two essential parts: an algorithm for enciphering and deciphering and a "key" that consists of information to be combined with the plaintext according to the dictates of the algorithm. In any modern encryption system, the algorithm is assumed to be known to an opponent, and the security of the system rests entirely in the secrecy of the key.

Our goal is to translate the language of the plaintext to a new language that cannot convey meaning without the additional information in the key. Those familiar with the concept of "entropy" in physics may be surprised to learn that it is also useful in information theory and cryptography. Entropy is a measure of the amount of disorder in a physical system or the relative absence of information in a communication system.

A natural language such as English has a low entropy because of its redundancies and statistical regularities. Even if many of the characters in a sentence are missing or garbled, we can usually make a good guess as to its meaning. In contrast, we want the language of our ciphertext to have as high an entropy as possible; ideally, it should be utterly random.

Our guiding principle is that we must increase the uncertainty of the cryptanalysts as much as possible. Their uncertainty should be so great that they cannot make any meaningful statements about the plaintext after examining the ciphertext; they must be just as uncertain about the key, even if they have the plaintext itself with the corresponding ciphertext (in practice, it is impossible to keep all plaintext out of their hands).

A prime consideration in the security of an encryption system is the length of the key. If a short key (short compared with the length of the plaintext) is used, the statistical properties of the language will begin to "show through" in the ciphertext as the key is used over and over; cryptanalysts will be able to derive the key if they have enough ciphertext to work with. On the other hand, we want a relatively short key so that it can be easily stored or even remembered by a human.

One way to meet these conflicting requirements is to use a short key to generate (or cause to be generated) an internal key of greater length, which is then used for the actual encryption. The trade-off here is that the long key must be generated by software, and it would seem that any key-generating software could be worked backwards to produce the rest of a key sequence if part of it is known. We will see that this is not necessarily true.

The other important fact about the keys is that we need a huge number of them. If our system allows only 10,000 different keys, for example, it is not secure. Our opponents could try every possible key in a reasonable amount of time. Fortunately, methods of generating a long internal key from a short external key also lend themselves to the production of a large number of different keys.

Thus, we want our system to be strong, fast, reasonably easy to code, and flexible enough to permit us to trade off speed for security or the opposite. The system we propose should satisfy the needs of the individual user who is concerned primarily with the storage of information and not with its high-speed transmission in large volumes. We assume also that the encryption is being carried out on the individual's own computer and not by a program running on some remote system.

Some Competing Systems

With these general considerations in mind, let's choose a specific encryption scheme for small systems. Three candidates come to mind: the government-sponsored Data Encryption Standard (DES), the new public key systems, or some approach to the infinite key cipher.

The DES is a product cipher using 16 stages of permutation and substitution on blocks of 64 bits each. The permutation tables are fixed, and the substitutions are determined by bits from a 56-bit key.² This short key has caused some experts to question the security of DES.³ Also, because the system was originally intended for hardware implementation, it would be slow in software.

We are inclined to reject DES merely because it is a standard. A standard cannot be changed for convenience—to use, say, only part of the transformations to speed up processing or a longer key for additional security. The individual user is better off without a standard, since breaking an unknown custom-built cipher presents greater difficulties for the cryptanalyst.

The public key cipher is an interesting new development that shows potential for making other encryption systems obsolete. It takes its name from the fact that the key information is divided into two parts, one of which can be made public. A person with the public key can encipher messages, but only someone with the private key can decipher them.⁴

All of the public key systems rely on certain functions for which the inverse is very difficult to compute without the information in the private key. These schemes do not appear to be practical for microcomputers, at least for 8-bit ma-

chines, if their strength is to be fully exploited.⁵ One variety of public key system, in fact, has been broken by solving its enciphering function,⁶ but this is no reflection on other systems, such as the RSA scheme,⁷ which use different enciphering functions.

We believe that the most practical encryption method for small systems is a product cipher in which the substitution mechanism as nearly as possible approximates the infinite key cipher.

The infinite key cipher, also known as the Vernam system or the "one-time pad," is simple in concept. We first generate a key that is random and at least the same length as our message. Then, for each byte of plaintext, we add the corresponding byte of the key to give the ciphertext. By "add" we mean some reversible operation; the usual choice is the exclusive-or.

A little reflection will show that, given a random key at least the size of the plaintext (i.e., "infinite" with respect to the plaintext because it is never repeated), the resulting cipher is unbreakable, even in principle.8

This scheme is in use today for the most secret government communications, but it presents a serious practical problem by requiring not only a long random key for each message but also that the lengthy key be sent to the recipient. Thus, the ideal infinite key system is not practical for large volumes of message traffic.

A Practical Infinite Key Cipher

We can design a substitution method that will approximate an infinite key by generating the key as we go. Of course, any deterministic method of generating a key has, in principle, some regularity that an analyst might exploit. However, if we make the "work factor" for breaking our system so high that it is impractical for our opponents to do so, then it is irrelevant that the system may be less strong than the ideal.

What constitutes an adequate work factor depends essentially on the number of uncertainties the cryptanalysts must resolve before they can derive plaintext or a key. In these days of constantly improving computers, that number should probably exceed 2¹²⁸.

It is easy to quantify the work factor if we are talking about exhaustive key trial, but few modern ciphers are likely to be broken by key trial since it is too easy to make the key prohibitively large. Most likely they will be broken because of internal periodicities and a subtle dependency of output on input; these give the cryptanalysts enough information to reduce their uncertainty by orders of magnitude.

Since increasing the work factor for the cryptanalysts also increases it for the encryption program, we want to build a modular system. That way we can add or remove transformation functions to trade off time for security, thus meeting whatever threat we reasonably expect.

We start with the problem of key length. How can we obtain a key effectively as long as the plaintext without having to actually specify it, byte by byte? The answer is key expansion.

Suppose we have three short keys, SECRECY (length 7), COMPUTER (length 9), and MATHEMATICS (length 11). We obtain a longer key by the exclusive-or of these three together, byte by byte. That is, S XOR C XOR M equals the

first key byte; E XOR O XOR A equals the second key byte, and so on. If we reach the end of a short key, we start over at its first character. If the key lengths are relatively prime (that is, they have no common divisor greater than one), then the period of the composite key is equal to the product of their periods $(7 \times 9 \times 11 = 693)$.

Obviously, we can generate a composite key of any length we want by increasing the length of the keys and by increasing their number. A useful article by Sclawy discusses this method and describes a version of it for ASCII files in BASIC.⁹

Key expansion is not a free ride, however; the effective length of the composite key, from the cryptanalytic point of view, is actually not as great as the product of the short keys because periodicities may appear in the key stream due to repeated traversal of the short keys. We can alleviate this weakness by simply increasing the size and number of our short keys. We can also obscure these periodicities by adding transposition.

Readers familiar with the history of cryptography may notice a resemblance between our short keys and the rotors of the mechanical cipher machines of World War II, such as the German ENIGMA. The rotor devices are relatively strong, but they can be broken, and we have taken their

			CIEVE SERVI		
ROTORO LEN = 19	ROTOR1 LEN = 23	ROTOR2 LEN = 31	PERM		
A	E	C	5		
V	W X	É	1		
C	P	ī	3		
X	A	ž	4		
L. E.	BOOK LOAD OF ST	Q	2		
AND ELAN					
Q	W <-START1	E			
T	Y	U			
I A	0 S	P D			
F	G	H			
J <- STARTO	K				
_					
M	L L	K			
J	Н	G			
F	D	S			
A	M	N			
B	V Z	C			
^	E-1 2 E				
E	THE WILLIAM STATES	Q			
= splice	R	Н			
A	X	M			
V =	G	н			
M	D	J. J. Commission of the commis			
C	= splice E	A			
L L	W W	I			
	X	Ĥ			
数 " 整 " 整 " 整 等 。	P	G			
· · · · · · · · · · · · · · · · · · ·	A	E			
	L.	W			
		M			
		A			
		= splice			
		С			
		A			
		E			
(1) (1) (1) (1) (1) (1) (1) (1) (1) (1)		I			
		Z Q <- START2			
START POSITIONS SHOWN AFTER ENCRYPTION OF 5 BLOCKS					
	Figure 1				

weaknesses into account in designing our system. Henceforth, we will usually refer to the short internal keys of our system as "rotors."

We now construct a toy encryption system based on this principle. If we also want to add permutation to our cipher, we must choose some unit or "block" of plaintext over which the permutation will take place. Let's say we choose a six-character block. Then we can also perform our substitutions over the same number of characters, advancing the rotor pointers by six after each block is processed.

Figure 1 (below) shows the tables involved. Note the three rotors carefully. They are of relatively prime lengths (19, 23, and 31), but the first six characters of each rotor of length n have been spliced onto the end of the rotor at position n + 1.

Thus, we simply add six to the rotor pointer after each block, instead of after every character. This arithmetic is modulo the rotor length, so we reset the pointers to the beginning of the rotor after the sum of the pointer plus six exceeds the rotor length. We call the rotor pointers the "starts," since they point to the starting position for the substitution.

Note also the table of numbers in 0..5 called "PERM." This is the permutation table, and it determines how the characters in the block are to be transposed. In this example, the number 5 is in the 0th position, so character 5 in the block goes to the 0th place; 0 is in the 1st position, so character 0 goes to the 1st position, and so on until arriving at the permuted result. Here, if the input block is ABCDEF, then after permutation the block will be FABDEC.

Leaving aside for the moment how PERM and the rotors are constructed, we can sketch a pseudo-code outline of our enciphering function in Listing One (page 54). To decipher, we submit the ciphertext to substitution and then perform the inverse permutation. Note that deciphering is not the inverse of enciphering; we must know which way we are going.

With a larger block size and a few more rotors, this is essentially the ENIGMA machine. It is actually a little stronger, since ENIGMA applied the inverse permutation after the substitutions, which made deciphering the inverse of enciphering.

Making the Practical Cipher Strong

Let's turn this toy system into a strong one. The most obvious enhancements are an increase in the number of rotors and the addition of more stages of permutation, consistent with a reasonable processing time. We chose six rotors and two stages of permutation for our system. The permutation occurs after the second and fourth substitutions.

Next, we determine that our system will operate on bits instead of bytes, for both substitution and permutation. (This is called "fractionation," and ciphers using it have historically been strong ones.) We take 256 bits (32 bytes) as our block size and perform all substitutions and permutations on such blocks.

We also want our rotors to be relatively large to achieve a long period, so we choose rotors whose lengths, in bits, are prime numbers less than 2048 (256 bytes). Specifically, we choose rotors of lengths 1789, 1787, 1783, 1777, 1759, and 1753, giving a composite period of 3.1 x 10¹⁹ bits. Choosing

prime numbers for the rotor lengths is an easy way to assure relative primality.

There is no reason to apply the same permutation to each block if PERM can take on a different arrangement for each block. As it turns out, a permutation table can cycle through a number of different arrangements before repeating itself. Our permutation table, now called BITPERM, consists of 256 randomly ordered bytes in 0..255, each byte value corresponding to a particular bit in the block. After each block, we will "cycle" BITPERM (see below) to a new position, allowing us to process 256 32-byte blocks before the same permutation is used again. This increases the period of the key stream to 7.9 x 10²¹ bits.

In our example, the rotors always step six characters at a time. We can easily provide for the rotors to step some arbitrary distance (modulo the rotor size). Our opponents now must contend with the fact that each rotor may be stepping anywhere from 1 bit to (rotor size–1) bits at a time. Also, we can change the order of the rotors for each session, giving 90 possible significant orderings.

We have drastically increased the uncertainties confronting our opponents, but we must add one further refinement. Even though our key stream is very long, we will use only the first tiny fraction of it—even for huge files. This is not only wasteful but dangerous. If the key stream is always set to the same starting point in its sequence, then identical plaintext files will produce identical ciphertext files. Even statistically similar files will produce similar ciphertext files. This gives the cryptanalysts more information about the key than we want them to know.

In principle, collecting enough such ciphertext will enable them eventually to reconstruct the rotors. To explode this possibility, we count the number of blocks we have enciphered and keep this information. Before each new session, we check this index and begin enciphering or deciphering from that point on. We write out the index as the first record of the ciphertext file and read it before deciphering; if enciphering, we save it in a separate file.

In our system, the index record is one CP/M record, with the first nine bytes reserved for the 72-bit index and the remaining bytes unused. The index thus describes the starting point for each rotor and one cycle of the permutation table. For example, the particular setting of the permutation table will be the residue of the index value divided by 256 (e.g., the index modulo 256). The starting point for the rotor of length 1789 will be the index value times the chosen step value modulo 1789, and so forth.

If two persons are corresponding, one could take all the blocks starting at block number 100 million, and the other could take all blocks starting at 1000 million. Even though they might send many identical message strings, the key stream used would always be different. In this way, we approach the requirement of the ideal infinite key cipher: that the key be used only once.

Note that we do not encipher the index record. We would have to do so at some agreed-upon index value to recover it, and this block would be nearly the same from message to message, again giving the cryptanalysts too much information. We follow the cryptographic principle: that which cannot be deeply hidden should be sent in the clear. This tells the

Use ALL the Power of Your MS-DOS, IBM PC-DOS, or CP/M-80 System with UNIX-Style Carousel Tools



ch "CP/M" "MS-DOS" <doc >newdoc diff newdoc doc I more ed newdoc kwic newdoc I sortmrg I uniq I unrot >index make -f makdoc ndx

Carousel Tools and Carousel ToolKits are trademarks of Carousel MicroTools, Inc. CP/M is a trademark of Digital Research; IBM is a trademark of International Business Machines; MS is a trademark of MicroSoft; UNIX is a trademark of Bell Laboratories.

CAROUSEL TOOLS are a proven set of over 50 programs designed to be used with pipes, redirected I/O and scripts. In the style of UNIX each Tool does one thing well, and the Tools can be used together to do more complex tasks.

YOU ACCOMPLISH MORE using Carousel Tools: better programming and documentation support, simpler data and file housekeeping, more general file handling.

TOOLS FOR PC/MS-DOS 2.x AND CP/M-80 are available now. The DOS ToolKit is \$149. The CP/M ToolKit is \$249 and includes a *shell* to provide pipes, redirected I/O, and scripts. Source code is available for \$100 more.

ORDER YOUR TOOLKIT TODAY.



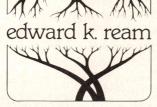


CALL OR WRITE:



609 Kearney Street, El Cerrito, CA 94530 (415) 528-1300

RED



FULL SCREEN EDITOR with

FULL SOURCE CODE in C for

CP/M 68K or CP/M 80

- RED is a powerful yet simple full screen text editor for both programmers and writers.
- RED features 17 commands including block move, block copy, search and substitute.
- RED comes with *full* source code in standard C. RED works as is with the BDS C, Aztec CII and Digital Research Compilers.
- RED supports *all* features of your terminal. You tailor RED to your terminal with an easy-to-use configuration program.
- RED handles files as large as your disk.
- RED is guaranteed. If for any reason you are not satisfied with RED, your money will be refunded promptly.

Price: \$95.

Call today for more valuable information: (608) 231-2952

To order, send a check or money order to:

Edward K. Ream

1850 Summit Avenue

Madison, Wisconsin 53705

Your order will be mailed to you within one week. Sorry, I can not handle credit cards. Please do not send purchase orders unless a check is included. RED is distributed only on 8 inch CP/M format disks, but other disk formats are available through third parties.

Dealer inquiries invited.

cryptanalysts nothing they would not know if we did not use an index, since they would then know from the algorithm itself what the starting points were.

Constructing the Keys

This design implies that we must construct six random tables of bits for the rotors, with the initial 256 bits spliced onto the end of each rotor. (We designed our rotors so that the rotor, plus the splice, occupies 256 bytes or less.) We must also construct the permutation table of 256 random bytes.

In a sense, these tables constitute our key, since the key stream used for encryption is derived from them. It is obviously impractical for us to enter the tables each time we wish to send or read a message. We would prefer to enter some manageable "user key" that would cause the tables to be generated. We could, of course, have no user key at all and just make the tables a fixed part of the algorithm. But eventually the tables would be compromised, and they are not easily changed.

A convenient solution, which we carefully considered, would be to use a software random-number generator to create the tables. The user key could then be the seed for this generator. Common ways of generating random numbers, such as the linear congruential method, are constructed so that the next random number is generated by transforming the preceding random number. ¹⁰ If one part of the sequence is known or correctly guessed, then all of the sequence can be computed.

We believe that this can be circumvented by constructing a generator of very large word size, say 96 bits, and using only the most significant 8 bits for the output. We do not see how such a generator could be worked backwards, but a comment by Knuth in one of his exercises suggests that it can be, at least for smaller word sizes.¹¹

We finally decided to abandon linear random-number generators entirely when we realized that we had a good nonlinear generator in the encryption system itself. If we have already constructed a set of genuinely random tables, we can take the user key, or a part of it, as a plaintext block that we then encipher using the predetermined rotors. The enciphered block replaces the first 32 bytes of the first rotor, and we encipher the just-enciphered block again; this time, however, the first rotor is different, having just been altered. We place the newly enciphered block in the second 32 bytes of the rotor and continue until we have filled all six rotors and the permutation table, thus replacing them entirely with new random values.

This self-modifying feature makes the new tables, which we call the "pro tempore" tables, depend on the user key in a very complex way. The pro tempore tables are the ones actually used for enciphering and deciphering, after having been properly spliced.

To guarantee that the output of this process is random, we must be sure that the predetermined tables are also random. Here is a cheap but effective method of doing so. To create the permutation table, begin with a bag of lima beans, a fine-point indelible ink pen, a bowl, a fork, and a tablet of paper. Carefully number 256 beans with the digits 0 through 255. As you number the beans, lay them out systematically before you on a flat surface in 16 rows of 16 columns each. This way

you can check that all numbers from 0 to 255 are present and that no numbers are duplicated. Place the numbered beans in the bowl and stir well with the fork. Without looking in the bowl, select beans one by one and record their numbers on the pad. Do not put the beans back in the bowl. The resulting table is the predetermined permutation table, which we call RANDP in our program.

The procedure for constructing the rotors is similar, but this time you must mark 64 beans with 1 on one side and 0 on the other, like the heads and tails of a coin. Stir these 64 beans with the fork and, again without looking, draw them out one by one and place them in a row. Record the resulting bit pattern and put the beans back in the bowl. Stir well again and repeat the process until you have all of the bits for the rotor in question.

Take the first 256 bits and write them down following the rotor bits. Divide the bit stream up into blocks of 8 bits and compute the hexadecimal value of each. This will give you a table of hex values that you can enter with an editor. Repeat this process until you have constructed all six rotors. This is tedious and requires meticulous work, but you must do it to guarantee genuinely random tables.

We decided upon three user keys, each of which may be between 1 and 75 ASCII characters in length. Since there are 95 printable ASCII characters, the user keys with the ASCII bias removed are considered as numbers in base 95 notation. We convert these numbers to binary (base 2) and pass them to the encryption functions.

We use one of the user keys as the initial plaintext in the table-generation process described above. The other user keys determine the stepping distances and the starting positions for the rotors during the generation phase and the stepping distances for the encryption session. The starting points for the session, of course, are determined by the index record.

The Program

We wrote our program in Z80 assembly language, using the Digital Research RMAC relocating assembler. This assembler uses the Intel mnemonics for the 8080 instructions and

macros for the Z80 instructions. Where the macros differ from the Zilog mnemonics, we supply a cross reference.

We would have preferred to write these routines in a high-level language, but the amount of time taken by the intensive bit manipulation precluded that. The listings that follow describe the encryption functions only; we omit the part of the program concerned with collecting the user keys and file I/O, since this code is straightforward.

First consider the module PARAMS (Listing Two, page 55). Here are the six rotors, the permutation tables, and storage for the starts, steps, rotor lengths (hereafter called the "rotor moduli" because of their use in computing the starts and steps), and user keys. The table I6, which determines the order in which the rotors are taken, will be shuffled into some random permutation. We extract 32 bytes of text from the file buffer, place it in BLOCK to be enciphered or deciphered, and then move it back to the buffer.

Note that most of these data areas must be assembled and linked so as to reside on a page boundary. For this extra effort, we earn an appreciable gain in speed in the permutation routines.

Note also the three tables BITPERM, RANDP, and I256 BITPERM is the table consulted by the bit permutation routines, but it is uninitialized at startup. BITPERM is constructed by a routine called CYCLE (described below) before generation of the pro tempore rotors begins, and it is reconstructed from RANDP after each block. After the initialization process, RANDP is replaced, just as the predetermined rotors are, but it is replaced by shuffling the ordered table I256 according to 256 random bytes generated at initialization. We cannot simply fill RANDP with random bytes, since it can contain no duplicates.

KEYGEN (Listing Three, page 58) controls the startup process just described. It first calls UPTACK (described below) to convert the user key to a binary number and then reconstructs the tables by repeated encipherment. After KEYGEN finishes its work, we call INIT (Listing Four, page 66), which uses the index information to set the permutation table and the rotor starting positions to their correct places for this particular point in the key stream. We are finally

Statistical Test of Key Generator

Average of 20 trials of 5120 bytes each

Normalized results Theoretical values

 Mean:
 .49785
 .50000

 First moment:
 .08351
 .08333

 Second moment:
 -.00003
 .00000

 Third moment:
 .01255
 .01250

Chi-square Statistic

Probablity: 99% 95% 75% 50% 25% 5% 1% Distribution: 187.2 205.5 233.4 254.3 276.5 310.3 335.9

Average of results: 258.9

Table

ready to encipher or decipher a file.

ENCIPHER (Listing Five, page 69) and DECIPHER (Listing Six, page 71) control the sequence of substitution and permutation on each block of text. Note that ENCIPHER takes the rotors in the order determined by I6. It calls EXTRACT to take 256 bits from the rotor in use, beginning at the current starting point, and to place them in the WORK common area. It calls SUBSF to perform the substitution on each block and calls PERMF to permute the bits in the block after the second and fourth substitution. Finally, it calls NXTBLK, which increments the index by one, steps the rotors to the next starting position, and sets BITPERM to its next permutation.

DECIPHER is similar. We take the rotors in the reverse order and call the inverse substitution and permutation functions, SUBSG and PERMG. (By mathematical convention, if "F" is a function, then "G" is its inverse.)

The substitution functions, SUBSF (Listing Seven, page 74) and SUBSG (Listing Eight, page 75) are simple. We chose to perform substitution modulo 2²⁵⁶ instead of modulo 2 (the exclusive-or); that is, we just add 32 bytes, discarding the final carry, for forward substitution and subtract 32 bytes without a final borrow for the reverse substitution.

With exclusive-or, cryptanalysts can narrow the possibilities for the key if they know the plaintext and ciphertext (i.e., if the ciphertext bit is 0 and the plaintext bit is 1, then the key bit must have been a 1, and so on). Addition mod 2²⁵⁶ spreads the uncertainty over the entire block, since the cryptanalysts don't know if there was a carry from byte to byte or not

PERMF (Listing Nine, page 76) and PERMG (Listing Ten, page 78) are the bit permutation routines, PERMF being the forward permutation and PERMG being its inverse. These routines may interest the reader quite apart from their cryptographic use in our program. Since hardware designers have not seen fit to give us machines with bit addressing, manipulations at the bit level are awkward at best. Even so, bit permutation is not difficult, if you define the problem correctly.

These routines address bits in one special case: a bit stream where bit 0, the first leftmost bit, is aligned on a byte boundary. We can imagine the bits in the stream as being indexed with subscripts from 0 to *n* from left to right. The general subscript is *i*, so that we can talk of the *i*th bit, and *i* is always a base 2 integer.

The trick used in these routines is to imagine the bit stream not as a vector but as a matrix consisting of x rows and y columns. Every byte is one row beginning with row 0. Every y is a bit within a row. There are 8 bits within each row subscripted from 0 to 7, left to right. To address the ith bit, we convert the vector index i to the matrix coordinates x, y.

APL fans will recognize this as the "encode" function with index origin zero. The actual computation is almost effortless because the modulus of the column coordinate y is an integral power of 2, so we merely shift the binary integer i to obtain x and y, giving us the byte and bit coordinates of the desired bit.

For illustration, let i be 115 decimal, which is 01110011 binary. Imagine this number now as an ordered pair (x,y) with a demarcation after the third bit from the right:

x y 01110:011

To the left of the demarcation is the binary value of x, and to the right is the value of y; in other words, the 15th byte, 3rd bit, corresponds to the 115th bit. Since most computers allow byte addressing, and even bit addressing by one means or another within a byte, this trick permits general bit addressing with a minimum of computation.

We now describe what we mean by "forward" and "inverse" permutations. Suppose BITPERM contains the byte values 57, 15, 21, ... 173. The forward permutation

57 15 21 ... 175 0 1 2 ... 255

means put the 57th bit in the 0th position, the 15th in the 1st, and so on to the 173rd bit in the 255th position. The inverse permutation means put the 0th bit in the 57th position, the 1st bit in the 15th, and so on.

The inverse permutation is about one-third faster than the forward permutation. This seems surprising until you realize that testing the bits for on or off is cheap in the inverse permutation but unavoidably expensive in the forward.

We have optimized PERMF and PERMG for speed, so they are not exactly models of structured programming. This is justified since most of the computation time required by TNT is spent in these two routines. Thus, we have required the operands for the permutation to be aligned on page boundaries to save instructions needed for byte addressing. This way, 8-bit registers alone can compute effective addresses. Also, we reach subroutines by jumps instead of by calls and returns.

CYCLE (Listing Eleven, page 81) is deceptively simple, but the concept behind it is abstract and perhaps unfamiliar to the reader; cryptographically it is critical. RANDP is a random permutation of the byte values 0..255, but RANDP is not a permutation list familiar to most programmers. It begins (in our example) 57, 15, 21, ...173. This does not mean "take the 57th element and put it in the 1st position." Rather, RANDP is a permutation in cyclic notation, denoted (57 15 21, ... 173). This cyclic notation means "take the 15th element and put it in the 57th position, then take the 21st element and put it in the 15th, and so on until you take the 57th element and put it in the 173rd position."

For simplicity, and at the expense of some cryptographic strength, we have chosen one cycle of degree 256—that is what CYCLE is designed for. But CYCLE could be extended to handle N cycles of different degrees. Some good two-cycle lengths for TNT are 148 and 107, 167 and 87, and so on. These lengths are primes whose sum is 256. A good four-cycle sequence is 83 73 53 47. Again, these are primes that add up to 256.

Were we to use two or more cycles, say m and n, against RANDP, we would take the first m elements of RANDP as the first cycle, the next n elements as the second, and so on. For example, if our cyclic sequence were 83 73 53 47, the first 83 elements would be the first cycle, the next 73 elements the second, the next 53 the third, and the last 47 the fourth.

C Programmers: Program three times faster with Instant-C™

Instant-C™ is an optimizing interpreter for C that makes programming three or more times faster. It eliminates the time wasted by compilers. Many repetitive tasks are automated to make programming less tedious.

- Two seconds elasped time from completion of editing to execution.
- Symbolic debugging; single step by statement.
- Compiled execution speed; 40 times faster than interpreted Basic.
- Full-screen editor integrated with compiler: compile errors set cursor to trouble spot.
- Directly generates .EXE or .CMD files.
- Follows K & R—works with existing programs. Comprehensive standard C library with source.
- Integrated package; nothing else needed.
- Works under PC-DOS*, MS-DOS*, CP/M-86*.

More productivity, less frustration, better programs. Instant-C™ is \$500. Call or write for more info.

Rational

(617) 653-6194 P.O. Box 480 Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), PC-DOS (IBM), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

Circle no. 53 on reader service card.

C COMPILER

- FULL C
- UNIX* Ver. 7 COMPATABILITY
- NO ROYALTIES ON GENERATED CODE
- GENERATED CODE IS REENTRANT
- C AND ASSEMBLY SOURCE MAY BE INTERMIXED
- UPGRADES & SUPPORT FOR 1 YEAR
- C SOURCE AVAILABLE FOR \$250000

HOST	6809 TARGET	PDP-11*/LSI-11* TARGET	8080/(Z80) TARGET	8088/8086 TARGET
FLEX*/UNIFLEX* OS-9*	\$200.00 WINOT \$350.00 WIN HOAT	500.00	500.00	500.00
RT-11*/RSX-11* PDP-11*	500.00	200.00 WIHOUT 350.00 WIH	500.00	500.00
CP/M* 8080/(Z80)	500.00	500.00	350.00 WILLIAM 350.00	500.00
PCDOS*/CP/M86* 8088/8086	500.00	500.00	500.00	200.00 WITHOUT 350.00 WITH HOAT

*PCDOS is a trademark of IBM Corp. MSDOS is a trademark of MICROSOFT. UNIX is a trademark of BELL LABS. RT-11/RSX-11/PDP-11 is a trademark of digital Equipment Corporation. FLEX/UNIFLEX is a trademark of Technical Systems consultants. CP/M and CP/M86 are trademarks of Digital Research. OS-9 is a trademark of Microware & Motorola

408-275-1659

TELECON SYSTEMS

1155 Meridian Avenue, Suite 218 San Jose, California 95125

Circle no. 64 on reader service card

ix Times Faster!

Super Fast Z80 Assembly Language Development Package

Z80ASM

- Complete Zilog Mnemonic set
- Full Macro facility
- Plain English error messages
- One or two pass operation
- Over 6000 lines/minute
- Supports nested **INCLUDE** files
- Allows external bytes. words, and expressions (EXT1 * EXT2)
- Labels significant to 16 characters even on externals (SLR Format Only)
- Integral cross-reference
- Upper/lower case optionally significant

- Conditional assembly
- Assemble code for execution at another address (PHASE & DEPHASE)
- Generates COM, HEX, or REL files
- COM files may start at other than 100H
- REL files may be in Microsoft format or SLR format
- Separate PROG, DATA & COMMON address spaces
- Accepts symbol definitions from the console
- includes TIME and DATE in listing (CP/M Plus Only)

Flexible listing facility

For more information or to order, call:

1-800-833-3061

In PA, (412) 282-0864 Or write: SLR SYSTEMS

1622 North Main Street, Butler, Pennsylvania 16001

SLRNK

- Links any combination of SLR format and Microsoft format REL
- One or two pass operation allows output files up to 64K
- Generates HEX or COM
- User may specify PROG. DATA, and COMMON loading addresses
- COM may start at other than 100H
- HEX files do not fill empty address space.
- Generate inter-module cross-reference and load map
- Save symbol table to disk in REL format for use in overlay generation
- Declare entry points from console
- The FASTEST Microsoft Compatible Linker available



- Complete Package Includes: Z80ASM, SLRNK, SLRIB - Librarian and Manual for just \$199.99. Manual only, \$30.
- Most formats available for Z80 CP/M, CDOS, & TURBODOS
- Terms: add \$3 shipping US, others \$7. PA add 6% sales tax

L R___Systems.

Why use cyclic notation? The cryptographic advantages in control and key length are great. The "order" of a permutation is defined as the least common multiple of the degrees of its cycles. The order means how many times the same permutation can be used to repeatedly permute something before that something permutes back to its starting arrangement. That is the whole idea cryptographically: we want to use the same thing (a permutation) repeatedly, but each time we want a different result. We also need to quickly compute what the Nth arrangement will be so that we can continue encryption of the next file where we left off from the previous file. Cyclic notation is perfect for this.

CYCLE uses RANDP to generate 256 different BITPERMs. If we used the cycles 149 and 107, we would generate 15,943 BITPERMs. By using more permutations in TNT, with well-chosen cycles, and staggering them, we could achieve thorough bit mixing with long periods.

We have restricted CYCLE to handle only one cycle of 256 elements, since we can quickly compute the position of the cycle by taking the index mod 256, which is just the least significant byte of the index. More time would be needed if we had to compute the residue of several prime cycles. Once we get the cycle position, we use it to shuffle RANDP into BITPERM. If the result is 0, then byte 57 (the 0th byte) of RANDP goes to byte 57 of BITPERM, byte 15 of RANDP goes to byte 15 of BITPERM, and so on. If the result is 1, then byte 15 (the 1st byte) of RANDP goes to byte 57 of BITPERM, byte 21 of RANDP goes to byte 15 of BITPERM, and so on.

UPTACK (Listing Twelve, page 82) converts a number represented in one base to a base 2 number. ("Uptack" is the informal name for the APL "decode" function.) Its argu-

ment is a byte string whose elements represent the number n in any base from 2 to 255. For example, suppose n is a base 10 number consisting of m digits. Then the argument is m bytes consisting of binary values ranging from 0 to 9. UP-TACK's second argument is the base. In TNT the base is always 95, but we have written the routine to handle the general case.

The algorithm is simple number theory. First zero an accumulator. Then add the most significant digit to it. If there is another digit of n, multiply the accumulator by the base and add the next digit in. Continue until the digits of n are exhausted.

UPTACK has a subtle purpose: it is meant to make the use and maintenance of the user keys easy. Key maintenance and security are dependent on the user's resources. The most secure keys are those that are utterly random, but such keys cannot possibly be remembered and so must be recorded. This means that the user must provide physical security to guard the keys, an expensive business possible for governments and large corporations but not for the average citizen. Individual users cannot have recorded keys and still feel secure since such keys can be stolen or seized under legal process.

We therefore want keys that can be easily memorized. Any key that can be memorized, however, tends to be cryptographically weak; your opponent may guess it or find it by exhaustive search. Suppose you use an English word of eight characters or fewer. It would be easy to try all such words from a dictionary in a reasonable amount of time.

A good user key, then, should be long, have no connection with you, and not be available to the opponent in any published work. Poetry is fine, but compose it yourself! UPTACK

Theoretical values

Statistical Test of Ciphertext

Average of 15 trials of 5120 bytes each

Mean:	.50102	.50000
Second moment:	.08327	.08333
Third moment:	.00016	.00000
Fourth moment:	.01247	.01250

Normalized results

Chi-Square Statistic

Probability: 99% 95% 75% 50% 25% 5% 1% Distribution: 187.2 205.5 233.4 254.3 276.5 310.3 335.9

Average of results: 254.2 Range: 206.8 to 292.2

Running Times

Initialization (including read of index):
 4.8 seconds

Enciphering (exclusive of disk i/o) for 31 CP/M records: 6.5 seconds, or 610 bytes/second

Deciphering (exclusive of disk i/o) for 31 CP/M records: 5.0 seconds, or 793 bytes/second

Figure 2

helps out here because it allows you to think of a key in human terms, even though it is just a random bit stream to the encryption program. Protection of the user keys is so important that most of the TNT code is occupied with it.

The other support routines, not shown, are:

- (1) EXTRACT, which extracts 256 bits from a rotor beginning with the nth bit to the nth + 256. On call, HL addresses the bit stream input on a page boundary, and DE holds n. On return, HL addresses the extracted bits aligned on a byte boundary in the WORK common area.
- (2) RESIDUE, which returns the residue of an 88-bit number divided by a 16-bit number. On call, DE holds the divisor, and HL addresses the dividend. On return, DE holds the result.
- (3) SHUFFLE, which sets up the permutation table, BIT-PERM, by shuffling the ordered array, I256. On call, BC addresses RANDP, DE addresses the workspace BITPERM, and HL addresses I256. On return, DE addresses BITPERM.
- (4) SPLICE, which splices the first 256 bits of a rotor onto the rotor after the n-1 bit of the rotor. On call, BC holds n (the rotor size), and HL addresses the rotor.
- (5) NXTBLK, which prepares the program for encryption of the next block. It advances the index value by one, advances each start by its corresponding step, and sets BIT-PERM to its next cycle.
- (6) RIPLO, which merely sets the WORK common area to zero.

Conclusion

At the beginning of this project we wondered whether a program requiring so much bit manipulation would be practical on an 8-bit microprocessor. We are pleased to report that the program runs rapidly and could probably be strengthened further without an unreasonable increase in processing time.

Figure 2 (page 52) summarizes the results with TNT running on a 4-MHz Z80. About half of the total time is used by the bit permutation routines. We also tested the ciphertext files for statistical randomness, using the methods suggested by Ruckdeschel. Figure 2 shows the results of several trials. The output appears to be completely random and independent of the input, even with plaintext files containing only zeroes or only ones. You must understand that a random output is only a *necessary* requirement for a strong encryption system, not a sufficient one, but we are delighted with the results anyway.

We feel we have satisfied the basic requirements for a strong encryption system:

- (1) A large key space
- (2) A long, practically infinite key period
- (3) A random key stream and a random ciphertext
- (4) No apparent dependency of output on input
- (5) A good combination of substitution and permutation
- (6) A user key that is convenient but not easily compromised We can think of several ways TNT could be improved. Huffman encoding of the plaintext files before enciphering would provide a substantial increase in speed for large files. For greater cryptographic strength, consider the improvements to the permutation routines suggested in the discussion of CYCLE. More rotors are always good, if you can stand the extra time. With a 16-bit machine, the possibilities

get even more interesting. If you decide to implement TNT yourself, construct your own tables.

For readers interested in seeing TNT run, we will send a standard 8-inch single-density disk containing the object code, all source files, and documentation to those who send \$25. It is traditional in introducing a new encryption system to issue a challenge cipher, and we include one on the disk. Our plaintext is an excerpt from a news story discussing government threats that persons interested in cryptography had better cooperate with the intelligence agencies or else. 14 We present the corresponding ciphertext, which, of course, includes the index record, and we challenge the reader to solve for the pro tempore rotors and RANDP. For a bonus, determine the natural language user keys. This provides you with everything a cryptanalyst could hope for, but if TNT cannot resist solution here, it will not have much practical value. We welcome your comments and suggestions.

References

- 1. Shannon, C., "Communication Theory of Secrecy Systems," *Bell System Technical Journal* (October 1949): 656-715.
- 2. Katzan, H., The Standard Data Encryption Algorithm, Petrocelli Books, 1977.
- 3. Diffie, W., and Hellman, M., "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *Computer 10* (June 1977): 74-84.
- 4. Hellman, M., "The Mathematics of Public Key Cryptography," *Scientific American*, Vol. 241, No. 16 (August 1979): 146-157.
- 5. Smith, J., "Public Key Cryptography," *Byte* (January 1983): 198-218.
- 6. "Adelman and Apple II Bust Public-Key Crypto Code," *Infoworld*, October 4, 1982.
- 7. Rivest, R., et al., "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Communications of the ACM*, Vol. 21, No. 2 (February 1978): 120-126.
- 8. Feistel, H., "Cryptography and Computer Privacy," *Scientific American*, Vol. 228, No. 5 (May 1973): 15-24.
- 9. Sclawy, A., "CP/M Encryption Prescription," *Micro-computer* (November 1980): 42-46.
- 10. Knuth, D., *The Art of Computer Programming*, Vol. 2, 2nd ed., Addison-Wesley, 1981.
- 11. Knuth, Vol 2, page 177.
- 12. Knuth, D., *The Art of Computer Programming*, Vol. 1, 2nd ed., Addison-Wesley, 1973.
- 13. Ruckdeschel, F., "Testing a Random Number Generator," in *Numbers in Theory and Practice*, p. 121, Byte Publications, 1979.
- 14. "CIA Director Warns Scientists," Science, January 22, 1982.

(Listings begin on page 54)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service No. 194.

File Encryption (Text begins on page 44)

Listing One

```
PSEUDO-CODE OUTLINE OF SIMPLE ENCIPHERING FUNCTION.
*
                                                            ×
*
  WE ASSUME:
                                                            *
       1. BLOCK, PERM, ROTORS AND STARTS ARE EXTERNAL
*
          TO THIS PROCEDURE.
*
       2. BLOCK, PERM AND THE ROTORS ARE ALREADY
*
                                                            ×
          CONSTRUCTED. AND.
*
       3. THE STARTS ARE INITIALIZED.
*
                                                            *
*
****************
ENCIPHER:
   PROCEDURE (BLOCK);
   DECLARE
                                 /* THE INPUT TEXT */
                  CHAR.
       BLOCK[6]
                                  /* THE PERMUTATION TABLE */
                  CHAR.
       PERM[6]
                                  /* THE SPLICED ROTORS */
       ROTORO[25]
                  CHAR,
       ROTOR1[29] CHAR,
       ROTOR2[35]
                   CHAR.
                                 /* THE ROTOR LENGTHS */
                   INT INIT(19),
       LENO
                   INT INIT (23),
       LEN1
                   INT INIT (29),
       LEN2
                                  /* ROTOR STARTING POINTS */
                   INT.
       STARTO
                   INT.
       START1
                   INT,
       START2
                                 /# SCRATCH SPACE #/
                   CHAR,
        WORK[6]
                                  /* INDEX */
                   INT:
        I
    /* PERFORM THE PERMUTATION */
                                  /# COPY BLOCK TO WORK #/
    WORK = BLOCK;
    FOR I = 0 TO 5:
        WORK[I] = BLOCK[PERM[I]]; /* EXCHANGE EACH CHAR */
    END FOR;
                                 /* PLACE RESULT IN BLOCK */
    BLOCK = WORK:
    / ×
       PERFORM SUBSTITUTION WITH EACH ROTOR.
       HERE, "+" REPRESENTS ADDITION MODULO SOME N FOR
       THE 6 CHARACTERS; IF N = 2 THEN THIS IS THE
       EXCLUSIVE-OR.
    */
        BLOCK = BLOCK + ROTORO[STARTO];
        BLOCK = BLOCK + ROTOR1[START1];
        BLOCK = BLOCK + ROTOR2[START2];
     /* ADVANCE THE STARTING VALUES FOR NEXT BLOCK */
```

```
STARTO = (STARTO + 6) MOD LENO;
START1 = (START1 + 6) MOD LEN1;
START2 = (START2 + 6) MOD LEN2;
```

RETURN (BLOCK);

END ENCIPHER:

End Listing One

Listing Two

```
PARAMS:
             A block of storage for the various parameters
;
                                                                      -
ş
        used by the encryption program.
                                              This block must
        be linked so as to reside on a page boundary.
             Most of these areas are accessed by pointers, not
        by their common names; the "common" definition is only
         to allow page alignment.
DSEG
         COMMON
                 /BLOCK/
BLOCK
        DS
                 32
                                   ;holds the block of text to be
                                   :...transformed by the encryption
                                   :...process.
        DS
                 224
                                   ;skip to next page
        COMMON
                 /WORK/
WORK
        DS
                 33
                                   ; work space used by various
                                   :..routines
        DS
                 223
                                   ;skip to next page
                                   ;the following rotors consist of
                                   ;..bits generated by a process
                                   ;..known to be truly random
                                   ;..with replacement
        COMMON
                 /ROTORO/
ROTORO
        DB
                 107, 97, 132, 125, 141, 82, 167, 197, 158, 229, 166, 29, 66
        DB
                 179, 22, 139, 240, 90, 117, 254, 137, 142, 130, 224, 162, 53
        DB
                 155, 104, 114, 51, 142, 244, 23, 43, 146, 187, 189, 134, 42
        DB
                 49, 163, 91, 152, 141, 151, 67, 111, 136, 135, 230, 136, 120
        DB
                 31, 236, 131, 159, 69, 94, 173, 107, 66, 105, 22, 109, 2, 235
                 124, 232, 165, 228, 212, 45, 213, 141, 195, 13, 140, 149, 105
        DB
        DB
                 158, 165, 206, 91, 192, 210, 75, 110, 216, 175, 198, 250, 0
        DB
                 181, 46, 169, 208, 194, 114, 238, 88, 210, 250, 145, 50, 106
                 247, 128, 139, 61, 132, 230, 83, 7, 246, 66, 133, 244, 197, 46
        DB
        DB
                 103, 243, 212, 170, 39, 72, 157, 18, 118, 198, 88, 66, 187, 36
        DB
                 88, 239, 217, 232, 5, 131, 51, 199, 107, 88, 175, 166, 20, 58
                 255,81,204,70,83,150,109,210,120,165,248,38,49,180
        DB
        DB
                 112, 237, 148, 245, 107, 32, 23, 82, 130, 223, 124, 226, 228, 129
        DB
                 129, 61, 210, 56, 98, 52, 243, 176, 142, 219, 123, 9, 7, 25, 11
        DB
                 127, 23, 166, 109, 203, 139, 59, 208, 136, 23, 102, 116, 112
        DB
                 121, 237, 29, 9, 138, 122, 20, 92, 188, 31, 153, 79, 102, 52, 136
        DB
                 178, 165, 12, 221, 56, 254, 203, 91, 12, 35, 236, 106, 149, 62
        DB
                 44, 247, 45, 48, 234, 21, 152, 180, 95, 130, 211, 175, 244, 76
        DB
                 116, 23, 5, 17, 172, 219, 67, 145, 156, 119, 160
                                                              (continued on next page)
```

File Encryption (Listing continued, text begins on page 44) Listing Two

		;The remaining rotors are not
		shown completely. Of course,
		each has 256 random values.
	COMMON	/ROTOR1/
ROTOR1	DB	177, 238, 109, 128, 55, 20, 169, 122, 87, 56, 108, 47, 166
	DB	113, 212, 11, 81, 246, 185, 69, 33, 29, 103, 77, 135, 110
	DB	13, 133, 244, 206, 58, 129, 106, 62, 215, 40, 164, 35, 172, 233
	DB	176, 237, 199, 70, 18, 172, 21, 61, 255
	COMMON	/ROTOR2/
ROTOR2	DB	104, 166, 98, 251, 237, 91, 132, 51, 97, 57, 121, 212, 210, 235
	DB	13, 215, 104, 176, 75, 144, 87, 76, 42, 169, 197, 11, 224, 165
	•	
	-	0/ 454 70 474 450 05 07 170 07 107 75 247 156 119
	DB	96, 151, 32, 174, 152, 85, 83, 138, 23, 193, 75, 243, 156, 119
	DB	184,237
	COMMON	/ROTOR3/
ROTOR3	DB	178, 17, 130, 147, 234, 183, 242, 38, 221, 88, 255, 36, 234
ROTORS	DB	141, 236, 112, 126, 233, 175, 188, 188, 89, 22, 161, 151, 89
	DD	1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
	DB	249, 19, 110, 172, 127, 146, 117, 70, 246, 56, 63, 116, 215, 222
	DB	94, 44, 139, 80, 203, 172, 227, 41, 157, 231, 39, 237, 61, 169
	COMMON	/ROTOR4/
ROTOR4	DB	145, 99, 204, 93, 181, 0, 29, 191, 66, 104, 100, 107, 152
	DB	249, 180, 183, 190, 159, 84, 59, 206, 225, 60, 115, 203
	•	
	DB	157, 194, 120, 231, 150, 15, 20, 58, 164, 220, 1, 100, 141
	DB	55, 238, 238
	COMMON	(DOTODE /
FOTOPE	COMMON	/ROTOR5/ 45,76,195,139,23,99,34,35,233,59,81,46,240,236
ROTOR5	DB	138, 228, 25, 112, 181, 239, 189, 37, 232, 156, 188, 21, 178
	DB	138, 228, 23, 112, 181, 237, 187, 37, 232, 188, 188, 188
	DB	222, 146, 244, 78, 94, 10, 217, 33, 219, 203, 133, 133, 211, 38
	DB	58,139,20
		the following table describes
		byte values generated by a
		;process known to be truly random,
		;without replacement
	COMMON	/RANDP/
RANDP	DB	57, 15, 21, 125, 40, 138, 197, 120, 3, 74, 232, 251, 239
	DB	82, 49, 121, 208, 142, 170, 195, 198, 206, 237, 233, 66, 90
	DP	199, 154, 177, 56, 98, 217, 126, 97, 116, 210, 70, 222, 203, 43
	DB DB	11, 100, 157, 28, 79, 178, 6, 145, 155, 99, 181, 169, 253, 173
DITECT	COMMON	/BITPERM/ ;holds the permutation table
BITPERM	מעו	256
		D. Dall's I Avaist

```
; ordered array to shuffled
         COMMON
                  /1256/
                                     :..into BITPERM
1256
         DB
                  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
         DB
                  19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34
         .
         DB
                  243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254
         DB
                  255
                                     ;following areas need not be
                                     ;..aligned on page boundary
         COMMON
                  /16/
16
         DB
                  0,1,2,3,4,5
                                     ; ordered array to shuffle for
                                     ;..rotor choice
         COMMON
                  /GUARD/
GUARD
         DB
                  0.0
                                     guard bytes for multiply overflow
         COMMON
                  /INDEX/
                                     :..from INDEX
INDEX
         DS
                                     ; the file index, which counts the
                                     ;...number of blocks which have
                                     :..been encrypted. Initially it
                                    ;..holds a user key, which is used
                                     ;..to generate the tables for the
                                     : . session.
         COMMON
                  /STEPO/
STEPO
         DS
                  2
                                     ; the stepping distances for the
STEP1
                  2
         DS
                                     ;..rotors
STEP2
         DS
                  2
STEP3
         DS
                  2
STEP4
         DS
                  2
STEP5
         DS
                  2
         COMMON
                  /STARTO/
STARTO
         DS
                  2
                                     the starting bit for rotor
START1
         DS
                  2
                                    ... stepping and key extraction
                  2
START2
         DS
START3
         DS
                  2
START4
         DS
                  2
START5
         DS
                  2
         COMMON
                  /MODO/
                                    ;rotor moduli
MODO
         DW
                  1789
MOD1
         DW
                  1787
MOD2
         DW
                  1783
MOD3
         DW
                  1777
MOD4
         DW
                  1759
MOD5
         DW
                  1753
         COMMON
                  /LENKY1/
LENKY1
         DS
                                    stores length of user key entered
         COMMON
                  /KEY1/
KEY1
         DS
                  75
                                    user key with ascii bias stripped
         COMMON
                  /LENKY2/
LENKY2
         DS
         COMMON
                  /KEY2/
KEY2
         DS
                  75
         COMMON
                  /LENKY3/
LENKY3
         DS
         COMMON
                  /KEY3/
KEY3
                                                                     End Listing Two
         DS
                  75
         END
                                                           (Listing Three begins on next page)
```

File Encryption (Listing continued, text begins on page 44) Listing Three

```
÷
            Generate the pro tempore, or session keys, from
  KEYGEN:
        the user keys. The pro tempore keys are:
3
            the six rotors, ROTORO - ROTOR5,
-
            the permutation table, RANDP,
            the rotor steping distances, STEPO - STEP5,
            the rotor starting bits, STARTO - STARTS, and,
            I6, the ordering index for rotors, starts,
                steps and moduli.
                LENKY1 and KEY1 to derive a tempory index
        We use
                 and I6.
                LENKY2 and KEY2 to derive the rotors and the
                 starts, and,
                 LENKY3 and KEY3 to derive the steps and RANDP.
        Macro
                         Ziloa
        ====
                         ____
                         LD IX, nn
        LXIX nn
                         JR C, nn
        JRC
            nn
                         LD (IX+n),r
        STX
             r,n
                         INC IX
        INIX
        CSEG
        MACLIB
                 Z80
        PUBLIC
                 KEYGEN
                 UPTACK, RESIDUE, ENCIPHER, CYCLE, SHUFFLE, SPLICE
        EXTRN
        EXTRN
                 IOBUFF
KEYGEN:
        PUSH
                 PSW
        PUSH
                 B
        PUSH
                 D
        PUSH
                 H
                 LENKY1
                                  convert key1 to base 2
        LDA
                                  ;length in B for UPTACK
        MOV
                 B,A
                                  ; key is in base 95
                 A, 95
         IVM
        LXI
                 H, KEY1
                 UPTACK
         CALL
                                  ; then move converted number
                 D. KEY1
         LXI
                 B, 32
                                  ;..back to KEY1 space
         LXI
         LDIR
         LDA
                 LENKY2
                                  ;convert key2 in same manner
         MOV
                 B.A
         MVI
                 A. 95
         LXI
                 H, KEY2
                 UPTACK
         CALL
                 D, KEY2
         LXI
                 B, 32
         LXI
         LDIR
```

```
LDA
                 LENKY3
                                  :..and key3 also
        MOV
                 B, A
        MVI
                 A, 95
        LXI
                 H. KEY3
        CALL
                 UPTACK
                 D, KEY3
        LXI
        LXI
                 B, 32
        IDIR
        LXI
                 D. GUARD+1
                                  ; clear INDEX for use by RESIDUE
        LXI
                 H, GUARD
        XRA
                 A
        VOM
                 M. A
        LXI
                 B, 10
        LDIR
                                  ;set up the rotor starting bits
        LXIX
                 STARTO
                                  ;..to whatever the user key
        LXI
                 H. KEY2+20
                                  ;.. specified, but adjusted to
        LXI
                 D, MODO
                                  :..0 <= START[[] < MOD[[]
        CALL
                 GETRES
                                  ; set up the rotor step values
        LXIX
                 STEPO
                                  ;..to whatever the user key
        LXI
                 H, KEY3+20
                                  :.. specified, similarily
        LXI
                 D. MODO
                                  ;..adjusted, but > 0
        CALL
                 GETRES
                                  ; if any step = 0, force it to 1
        LXI
                 H. STEPO
        IVM
                 B. 6
TO1LOOP:
        MOV
                 E.M
                                  ;get the two-byte step
        INX
                 H
                                  :.. to DE
        MOV
                 D, M
        MOV
                 A,E
                                  ;check it for zero
        ORA
                 D
        CZ
                 SETTO1
                                  ;set it to 1 if so
        INX
                 H
        DJNZ
                 TO1LOOP
                                  ;set up a temporary index
        LXI
                 D. INDEX
        LXI
                 H, KEY1+23
        LXI
                 B, 9
        LDIR
                                  ;set up BITPERM to Ith iteration
        LDA
                 INDEX+8
                                  ;..by index mod 256
        LXI
                 H, RANDP
        LXI
                 D. BITPERM
        CALL
                 CYCLE
           The encryption parameters are now set up per the
        ş
           user keys. We will use this setup as a random
           number generator to create the pro tempore rotors,
           permutation table, starts and steps actually used
                                                                     ;
           for the file encryption.
                                                                     ÷
```

File Encryption (Listing continued, text begins on page 44) Listing Three

	LXI LXI LXI LDIR LXI MVI	D, BLOCK H, KEY2 B, 32 D, ROTORO B, 48	;first, create the rotors ;by repeatedly enciphering ;second user key ;do 48 blocks to fill the rotors
		D, 40	, do 40 blocks to 1111 the 10to/s
FILLROT	ORS:	ENCIPHER	;encipher the block
	LXI	H, BLOCK	;and move it to the rotor
	PUSH LXI	B B,32	
	LDIR POP	В	;leaves DE = DE + 32
	DJNZ	FILLROTORS	
			;The rotors aren't completely ;fixed yet. This must wait
			<pre>;until I6 is generated, ;since it controls the order ;of rotors, etc.</pre>
	LXI	D, BLOCK	;next generate pro tempore RANDP
	LXI	H,KEY3 B,32	;move key3 to encryption block
	LDIR		
	LXI MVI	D, IOBUFF B, 8	;use i/o buffer as scratch space ;encipher 8 blocks
FILLRAN	inp.		
FILLRAN	CALL	ENCIPHER	;encipher the block
	LXI	H, BLOCK	;and move it to scratch area
	PUSH	B	
	LXI	В, 32	
	POP	В	
	DJNZ	FILLRANDP	
		W 7057	;call SHUFFLE to replace RANDP
	LXI	H, I 256 B, I OBUFF	address ordered table; address 256 random bytes
	LXI	D, RANDP	, address 250 random bytes
	CALL	SHUFFLE	now RANDP is different than
			;the permanent RANDP
			;now generate I6
	LXI	D, BLOCK	;move key1 to encryption block
	LXI	H,KEY1 B,32	
	LDIR		
	LXI	D, IOBUFF	;use i/o buffer as scratch space
	MVI	B,8	; do 8 blocks

TAKE CONTROL WITH ULTIMATE CONTROL

Don't you wish you could alter your word processor to fit your needs, move it to a new system, or upgrade it whenever you wanted?

Your problems are over. The **ULTIMATE CONTROL** Word Processing book contains:

- (1) a complete editor
- (2) Cross reference generator
- (3) Word count
- (4) Multi file corrections facilities
- (5) Text file organizer
- (6) complete formatter
- (7) all source code (most in Basic)
- (8) design document, installation guide and operator instructions
- (9) and much, much more.

ALL OF THIS IN A BOOK FOR JUST \$24.95. Ask for it at your local computer or book store, or write to:

MCDERMOTT COMPUTER SERVICES 12 Manor Haven Road Toronto, Ontario, M6A 2H9

Mail orders should include \$3.00 for shipping. Cheque, or money order (No C.O.D.),

Circle no. 36 on reader service card.

GREP in C The UNIX*regular expression recognizer

_MAIN

Wildcard expansion & pipes for Aztec C

All programs come with complete source code, in C. Price: \$35 each: \$50 together.

For more information or complete catalogue:

SOFTWARE ENGINEERING CONSULTANTS
P. O. BOX 5679
BERKELEY, CA 94705

(415) 548-6268

+ A trademark of Bell Laboratories

Circle no. 59 on reader service card.

HOW FAST WOULD THIS PROGRAM RUN IF IT WERE COMPILED USING YOUR PASCAL COMPILER?

PROGRAM SIEVE;
{ THE ERATOSTHENES' SIEVE BENCHMARK }

CONST SIZE = 8190; TYPE BYTE = 0..255; VAR I, PRIME, K, COUNT, ITER : INTEGER; FLAGS : ARRAY [0..SIZE] OF BOOLEAN;

BEGIN

WRITELN('START');
FOR ITEM := 1 TO 10 DO BEGIN

COUNT := 0;
FOR I := 0 TO SIZE DO FLAGS[I] := TRUE;
FOR I := 0 TO SIZE DO

IF FLAGS[I] THEN BEGIN
PRIME := | + | + | 3;
K := | + PRIME;
WHILE K <= SIZE DO BEGIN
FLAGS[K] := FALSE;
K := K + PRIME
END;
COUNT := COUNT + 1
END;
END;
WRITELN(COUNT, 'PRIMES')

Chances are, not as fast as it would if it were compiled using SBB Pascal.

As the following benchmarks show, SBB Pascal outperforms all other Pascal compilers for the PC in terms of speed, code size and .EXE file size:

	Execution Time (secs)	Code Size	EXE File Size
SBB Pascal	10.90	181	4736
MS-Pascal	11.70	229	27136
Pascal/MT+ 86	14.70	294	10752
Turbo Pascal	15.38	288	9029

Development Package \$350.00



Personal Use Compiler Package also available \$95.00

Call for free brochure with full benchmarks.

607/272-2807

Software Building Blocks, Inc. Post Office Box 119 Ithaca, New York 14851-0119

SBB Pascal is a trademark of Software Building Blocks, Inc. MS-Pascal is a trademark of Microsoft Corporation. Pascal/MT+ 86 is a trademark of Digital Research, Inc. Turbo Pascal is a trademark of Borland International.

File Encryption (Listing continued, text begins on page 44)

Listing Three

```
FILLI6:
                 ENCIPHER
                                   encipher block
        CALL
                                   ... and move it to scratch area
        LXI
                 H. BLOCK
        PUSH
                 B
                 B, 32
        LXI
        LDIR
        POP
                 B
                 FILL 16
        DJNZ
                                   :shuffle I256 to replace I6
        LXI
                 H, I256
                 B, IOBUFF
        LXI
                                   ;shuffled I256 goes here
         LXI
                 D, IOBUFF+256
         CALL
                 SHUFFLE
                                   ;now find the six bytes < 6
         LXI
                 D, 16
                 H. IOBUFF+256
         LXI
                                   ;loop control
         MVI
                 B. 6
SETI6:
                                   : get a random byte
         MOV
                 A, M
                                   ; is it < 6?
         CPI
                  6
                                   ; jump if so
                  IOK
         JRC
                                   try another
         INX
                 H
                  SETI6
         JMP
IOK:
                                   :save it in I6 if < 6
         STAX
                  D
                                   ;bump destination pointer
                  D
         INX
                                   ;bump source pointer
         INX
                  H
                  SETI6
         DJNZ
                                   ;finally, splice the rotors,
                                   ;...in the order for this session,
                                   :.. now that we have I6
                  H. ROTORO
         LXI
                                   count 6 rotors, C indexes rotors
         MVI
                  B. 6
DOSPLICE:
                  B
         PUSH
         PUSH
                  H
                  D, MODO
                                    ; address rotor moduli
         LXI
                  H, 16
         LXI
                                    get 16 value
                  A,M
         MOV
                                    * 2 for offset into moduli
         ADD
                  A
                                    ; add to DE
                  E
         ADD
         MOV
                  E,A
         JNC
                  DOSP1
         INR
                  D
 DOSP1:
                                    ;HL -> the modulus to use
         XCHG
         MOV
                  C, M
                                    ; get it to BC
          INX
                  H
         MOV
                  B, M
                                    recover rotor address
          POP
                                    ;splice the rotor
          CALL
                  SPLICE
                                    ;point to next rotor (HL+256)
          INR
                  H
```

```
POP
        DJNZ
                 DOSPLICE
                                   ; now each ROTOR[[6[]]] is of
                                   ;..length MOD[I6[I]], but in an
                                   ;..order which depends on the
                                   :..user key
        POP
                 H
                                   ;all done ...
        POP
                 D
        POP
                 B
        POP
                 PSW
        RET
                                   :GETRES: interface between KEYGEN
                                   ; .. and RESIDUE. IX -> destination,
                                   ;..HL -> source, DE -> mod, using
                                   :.. INDEX for dividend
GETRES:
        MVI
                 B, 6
                                   ;loop control
RESLOOP:
        PUSH
                 B
        PUSH
                 D
                 D, INDEX+7
        LXI
                                   ; move key bytes to INDEX
        LXI
                 B, 2
        LDIR
                                   ;now HL -> next source
        POP
                 D
        XCHG
                                   ;HL -> mod
        PUSH
                 D
        MOV
                 E,M
                                   ; load modulus to DE
        INX
                 H
        MOV
                 D. M
        INX
                 H
                                   ;HL -> next mod
        PUSH
                 H
        LXI
                 H. GUARD
        CALL
                 RESIDUE
                                   ; find the residue
        STX
                 E,0
                                   ;set destination to result
        STX
                 D, 1
        INXIX
                                   ;next destination
        INXIX
        POP
                 D
                                   ;-> next mod
        POP
                 H
                                   ;-> next source
        POP
                 B
                 RESLOOP
        DJNZ
        RET
                                   ; SETTO1:
                                             set step to 1 if req'd
SETTO1:
        MOV
                 M. A
                                   ;zero high byte (we know A is zero)
        INR
                 A
                                   ; make A = 1
        DCX
                 H
                                   ;-> low byte
        MOV
                 M. A
                                   ; store 1
        INX
                 H
                                   ;reset pointer
        RET
        COMMON
                 /BLOCK/
                                   ;the encryption block
BLOCK
        DS
                 32
                 /ROTORO/
        COMMON
                                  start of the rotor space
ROTORO
        DS
                 256
```

File Encryption (Listing continued, text begins on page 44) Listing Three

	COMMON	/RANDP/	;permutation table	
RANDP	DS	256		
	COMMON	/BITPERM/	;table used for bit permuta	tion
BITPERM		256		
	COMMON	/1256/	; ordered table to be shuffle	ed
1256	DS	256		
	COMMON	/16/	;table for permutation of re	otors
16	DS	6		
	COMMON	/GUARD/	: guard bytes for index space	-
GUARD	DS	2	igual o bytes for Index speed	
COLIND	COMMON	/INDEX/		
INDEX	DS	9		
	COMMON	/STEPO/	;storage for the steps begin	ns
STEPO	DS	2		
	COMMON	/STARTO/	;storage for the starts beg	ins
STARTO	DS	2		
	COMMON	/MODO/	;storage for rotor moduli b	egins
MODO	DS	2		
	COMMON	/LENKY1/	storage for user key lengt	he
LENKY1	DS	1	; storage for user key rengt	113
LEINNII	COMMON	/KEY1/	and user keys	
KEY1	DS	75		
	COMMON	/LENKY2/		
LENKY2	DS	1		
	COMMON	/KEY2/		
KEY2	DS	75		
	COMMON	/LENKY3/		
LENKY3	DS			
	COMMON	/KEY3/		
KEY3	DS	75		
	END			End Listing Thre
	CIATA			Life Libring IIII

(Listing Four begins on page 66)

Elegance
Power
Speed



€ Users' Group Supporting All C Users Box 287

Yates Center, KS 66783

Circle no. 16 on reader service card.

Now Your Computer Can See! \$295.00*

A total imaging system complete and ready for plug-and-go operation with your personal computer.

The MicronEye" offers selectable resolution modes of 256 x 128 and 128 x 64 with operating speeds up to 15 FPS. An electronic shutter is easily controlled by software or

manual functions, and the included sample programs allow you to continuously scan, freeze frame, frame store, frame compare, print and produce pictures in shades of grey from the moment you begin operation.

Only the MicronEye[™] uses the revolutionary IS32 OpticRAM[™] image sensor for automatic solid state image digitizing, with capability for greytone imaging through multiple scans. And with these features, the MicronEye[™] is perfectly suited for graphics input, robotics, text and pattern recognition, security, digitizing, automated process control and many other applications.

The MicronEye™ is available with immediate delivery for these computers: Apple II, IBM PC, Commodore 64 and the TRS-80CC (trademarks of

Apple Computer Inc., International Business Machines, Commodore Corp., and Tandy Corp. respectively).

Phone for MicronEye™ information on the Macintosh, TI PC and RS232 (trademarks of Apple Computer Inc. and Texas Instruments respectively.)

*(Add \$8,00 for shipping and handling [Federal Express Standard Air]; residents of the following states *must add sales tax*: AK, AZ, CA, CO, CT, FI, GA, IA, ID, IL, IN, LA, MA, MD, ME, MI, MN, NC, NE, NJ, NY, OH, PA, SC, TN, TX, UT, VA, VT, WA, WI.)

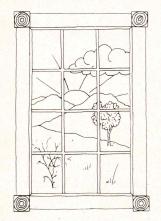
MICRON TECHNOLOGY, INC.

MicronEye"

"Bullet"

VISION SYSTEMS 2805 East Columbia Road Boise, Idaho 83706 (208) 383-4106 TWX 910-970-5973

Circle no. 39 on reader service card



WINDOWS FOR C™

FOR THE IBM PC + COMPATIBLES

Lattice C, DeSmet C C86, Microsoft C All versions

C Advanced Screen Management Made Easy

ADVANCED FEATURES

- Complete window system
- Unlimited windows and text files
- Nest and overlap windows
- Overlay, restore, and save windows
- Horizontal and vertical scrolling
- Word wrap, auto scroll
- Print windows
- Highlighting
- Fast screen changes
- · No snow, no flicker

WINDOWS++

Much more than a window display system, **Windows for C** is a video toolkit that simplifies all screen management tasks.

SIMPLIFY • IMPROVE

- Menus
- Help files
- Data screens
- Editors

ALL DISPLAYS

C SOURCE MODULES FOR

pop-up menus, multiple window displays, label printer, cursor control, text mode bar graphs.

Plus complete building block subroutines

DESIGNED FOR PORTABILITY

*Minimal dependence on IBM BIOS and 8086 ASM

FULL SOURCE AVAILABLE
NO ROYALTIES

Windows for C Demo disk and manual \$150

\$ 30

(applies toward purchase)

A PROFESSIONAL SOFTWARE TOOL FROM

CREATIVE SOLUTIONS

21 Elm Ave. Box D8, Richford, VT 05476

802-848-7738

Master Card & Visa Accepted Shipping \$2.50 VT residents add 4% tax

File Encryption (Listing continued, text begins on page 44)

```
Listing Four
```

```
-
   INIT: Initialize the index from the file index
3
        (if enciphering), or from ciphertext file, if
       deciphering. Also set BITPERM according to
-
       the index mod 256, and the rotor starting
       positions according to index and the steps.
       The index - 1 counts the the I - 1 block
       enciphered.
-
            This routine sets the pro tempore rotors
       and the bit permutation table to what they
3
       should be for enciphering the Ith block.
       Thus, no part of the key stream is used more
       than once.
       On call, HL -> encryption index
-
       Macros
                        Zilog
       _____
3
                       _____
       RARR M
                       RR (HL)
       JRC nn
                       JR C,nn
       RALR M
                       RL (HL)
       CSEG
       MACLIB Z80
       PUBLIC INIT
       EXTRN CYCLE, RESIDUE, RIPLO
INIT:
       PUSH
               PSW
       PUSH
               B
       PUSH
               D
       PUSH H
       LXI
               D. INDEX
                                :destination area
       LXI
               B, 9
       LDIR
                                ; INDEX = file or ciphertext index
       LXI
               D, BITPERM
       LXI
               H. RANDP
       LDA
               INDEX+8
       CALL
               CYCLE
                                ;BITPERM gets the permutation of
                                :.. RANDP per index mod 256
          Now set starting positions for rotors:
          DO I = 1 TO 6
              START[[6[]]] = (INDEX * STEP[[6[]])
                                [[I]6I]dOM bom
          END DO
       LXI
               H. 16
                               ;-> indexes of START, STEP, MOD
       MVI
              B, 6
                                ;loop control
```

```
INITO1:
                                   : get [6[]]
        MOV
                 A.M
        ADD
                 A
                                   : offset from STARTO
                                   ; save it
        STA
                 OFFSET
                                   :-> [6[]+1]
                 H
        INX
        PUSH
                                   ...and save for next loop
                 H
        LXI
                 D. STEPO
        ADD
                 E
        MOV
                 E.A
        JNC
                 INITO2
        INR
INITO2:
        CALL
                 MULT11
                                   ;multiply index * STEP[[6[]]]
        LXI
                 H. MODO
                                   :get MOD[[6[]] into DE
        LDA
                 OFFSET
                                   ... for RESIDUE
        ADD
                 L
        MOV
                 L.A
        JNC
                 INITO4
         INR
INITO4:
        MOV
                 E.M
        INX
                 H
        MOV
                 D.M
        LXI
                 H. WORK+11
                                   ;-> to 11 byte result of multiply
        CALL
                 RESIDUE
                                   return with DE = residue
        LXI
                 H. STARTO
                                   ; get START[[6[]]] to store
        LDA
                 OFFSET
                                   :..the residue
         ADD
                 1
                 L.A
        MOV
                 INITO6
         JNC
         INX
INITO6:
         MOV
                 M.E
                                   :HL -> START[[6[]]]
         INX
                 H
                                   :..store residue there
         MOV
                 M. D
         POP
                 H
                                   ;retrieve I6[I+1]
         DJNZ
                 INITO1
                                   :do for I = 1 to 6
         POP
                 H
         POP
                 D
         POP
                 B
         POP
                 PSW
         RET
                                   :MULT11: multiplies the value
                                   ...pointed to by DE by 11-byte
                                   ;..value in the WORK area
MULT11:
         PUSH
                 B
                                   ;save caller's loop control
                 B. 2
                                   ;copy multiplier for shifting
         LXI
         LXI
                 H, SHIFTER
         XCHG
         LDIR
                 RIPLO
         CALL
                                   :clear work area
         LXI
                 H, INDEX
                                   copy multiplicand into WORK+2
         LXI
                 D, WORK+2
         LXI
                 B. 9
         LDIR
```

File Encryption (Listing continued, text begins on page 44) Listing Four

```
MVI
                 B, 16
                                    ; shift 16 bits
MULTO1:
        LXI
                 H, SHIFTER+1
        ANA
                 A
                                    ;clear carry
        RARR
                 M
                                    ;test for add of partial product
         JZ
                 MSBISO
                                    ;...and possibly done
        DCX
        RARR
                 M
MULTO3:
         JRC
                  INTERADD
                                    ; add accum to partial result,
MULTO4:
                                    ...with more doubling to go
        PUSH
                 B
                 B, 11
        IVM
        LXI
                 H, WORK+10
         ANA
                 A
                                    ;clear carry
MULTO5:
        RALR
                 M
                                    ;double multiplicand by shifting
        DCX
                 H
        DJNZ
                 MULT05
                                    ;...for all 11 bytes
        POP
                 B
                                    ;restore loop control
        DJNZ
                 MULT01
                                    ; do for all bits in SHIFTER
        POP
                                    ;restore caller's loop control
        RET
INTERADD:
        PUSH
                 B
        LXI
                 B, 11
                                    ; add 11 bytes
        LXI
                 D. WORK+21
                                    ;-> partial result
                 H, WORK+10
        LXI
                                    ;-> doubled value
         ANA
                 A
                                    ; clear carry
INTADDO1:
        LDAX
                 D
        ADC
                 M
         STAX
                 D
         DCX
                 D
         DCX
                 H
         DJNZ
                  INTADD01
        POP
                 B
         DJNZ
                 MULT04
                                    ;do for all bits in shifter
        POP
                 B
        RET
FINALADD:
        MVI
                 B, 11
        LXI
                 D, WORK+21
        LXI
                 H, WORK+10
        ANA
                 A
                                    ; clear carry
FLADDO1:
        LDAX
                 D
        ADC
                 M
        STAX
                 D
```

```
DCX
                 D
        DCX
                 H
        DJNZ
                 FLADD01
        POP
                 B
        RET
MSBISO:
                                    ; arrive here if most signif byte
                                    ;..of shifter is 0
         INX
                                    ;..and test for least signif 0
         RARR
                  M
         JNZ
                  MULT03
         JRC
                  FINALADD
         POP
                  B
         RET
         DS
OFFSET
                  1
                  2
                                    ; multiplier
SHIFTER DS
         COMMON
                  /WORK/
WORK
         DS
                  33
         COMMON
                  /RANDP/
RANDP
         DS
                  256
         COMMON
                  /BITPERM/
BITPERM DS
                  256
         COMMON
                  /16/
16
         DS
                  6
         COMMON
                  /INDEX/
INDEX
         DS
         COMMON
                  /STARTO/
STARTO
         DS
         COMMON
                  /STEPO/
STEPO
         DS
         COMMON
                  /MODO/
MODO
                  2
         DS
         END
                                                                   End Listing Four
```

Listing Five

```
5
   ENCIPHER:
              Encipher one 32-byte block of text by
        substitution with the six rotors and permutation
        by BITPERM.
        Macros
                         Zilog
        _____
                         ____
                                                                    ;
;
                         SLA r
        SLAR r
        CSEG
        MACLIB
                Z80
        PUBLIC
                ENCIPHER
                EXTRACT, SUBSF, PERMF, NXTBLK
        EXTRN
```

(continued on next page)

File Encryption (Listing continued, text begins on page 44) Listing Five

ENCIPHE	D.		
ENCIPHE	PUSH	PSW	
	PUSH	В	
	PUSH	D	
	PUSH	Н	
	LXI	Н, 16	;address the vector which
			determines the order of
			:the rotors
	MVI	B, 6	;loop control
ELOOP:			
	MOV	D, M	;set DE to relative page of
	MVI	E,0	:.rotor tables
	INX	Н	for next rotor
	PUSH	Н	나 있는데, 맛데데, 가를 가장이를 하면 하면 하면 있다. 아이들이 많은 이렇게 하는데
		경영한 투성적인 시민생활 보는 집에 되고 먹는 맛이라고 모든 것이다.	;save I6 address
	LXI	H, ROTORO	;base of rotor tables
	DAD	D	;HL -> rotor to be used
	PUSH	Н	;save rotor address
	LXI	H,STARTO	;base of start addresses
	MOV	E,D	;set DE to relative word of starts
	SLAR	E	;E = E * 2
	MVI	D, 0	;now DE = index to correct start
	DAD	D	;HL -> start value to be used
	MOV	E,M	;get the start value to DE
	INX	Н	
	MOV	D, M	
	POP	н	;HL -> rotor
	CALL	EXTRACT	extract 256 bits from the rotor
	LXI	D, BLOCK	
	CALL	SUBSF	;perform the substitution
	MOV	A, B	;permute after second & fourth
	CPI	5	substitutions
	CZ	DOPERM	, substitutions
	CPI	3	
	CZ	DOPERM	
	POP	Н	;restore I6 pointer
	DJNZ	EL00P	;do for six rotors
	CALL	NXTBLK	; advance the index and step rotors
	POP	Н	
	POP	D	
	POP	В	
	POP	PSW	
	RET	LOW	
DOPERM:			;do a permutation on block
	XCHG		;HL -> BLOCK
	LXI	D, BITPERM	;address permutation table
	CALL	PERMF	;forward permutation
	RET		

```
COMMON
                 /BLOCK/
BLOCK
        DS
                 /16/
        COMMON
        DS
                 6
16
        COMMON
                 /STARTO/
STARTO
        DS
        COMMON
                /ROTORO/
                 256
ROTORO
        DS
                 /BITPERM/
        COMMON
                 256
BITPERM DS
        END
```

End Listing Five

Listing Six

```
;
   DECIPHER: Decipher one 32-byte block of text by
=
        substitution with the six rotors and permutation
        by BITPERM.
        Macros
                         Zilog
        _____
                         ____
                         SLA r
        SLAR r
        CSEG
        MACLIB
                Z80
        PUBLIC
                DECIPHER
                 EXTRACT, SUBSG, PERMG, NXTBLK
        EXTRN
DECIPHER:
                 PSW
        PUSH
        PUSH
                 B
        PUSH
                D
        PUSH
                H
        LXI
                H, I6+5
                                  ; address the vector which
                                  ;...determines the order of
                                  ;..the rotors
        IVM
                 B, 6
                                  ;loop control
DLOOP:
        MOV
                 D. M
                                  ;set DE to relative page of
        IVM
                 E, 0
                                  :..rotor tables
        DCX
                 H
                                  ; for next rotor
        PUSH
                                  :save I6 address
                                  ;base of rotor tables
                 H, ROTORO
        LXI
        DAD
                                  ;HL -> rotor to be used
                 D
        PUSH
                                  ; save rotor address
                 H
                                  ;base of start addresses
        LXI
                 H, STARTO
                                  ;set DE to relative word of starts
        MOV
                 E, D
        SLAR
                 E
                                  E = E * 2
                                  ;now DE = index to correct start
        MVI
                 D, 0
                                  ;HL -> start value to be used
        DAD
                 D
        MOV
                 E,M
                                  get the start value to DE
        INX
                 H
        MOV
                 D.M
                                                             (continued on next page)
```

File Encryption (Listing continued, text begins on page 44) Listing Six

	POP CALL LXI CALL	H EXTRACT D, BLOCK SUBSG	;HL -> rotor ;extract 256 bits from the rotor ;perform the substitution	
	MOV CPI CZ	A, B 5 DOPERM	;permute after second and fourth ;substitutions	
	CPI CZ POP	3 DOPERM H	;restore I6 pointer	
	DJNZ	DLOOP	;do for six rotors	
	CALL	NXTBLK	; advance the index and step rotors	
	POP POP POP POP RET	H D B PSW		
DOPERM:				
	XCHG LXI CALL RET	D,BITPERM PERMG	;HL -> BLOCK ;address permutation table	
BLOCK	COMMON DS COMMON	/BLOCK/ 32 /16/		
16	DS COMMON	6 /STARTO/		
STARTO	DS COMMON	2 /ROTORO/		
ROTORO	DS COMMON	256 /BITPERM/		
BITPERM	DS END	256	End Listing S	Six

(Listing Seven begins on page 74)

Zizzing zeren ergine en Fuge . . .

MicroMotion

MasterFORTH

It's here — the next generation of MicroMotion Forth.

- Meets all provisions, extensions and experimental proposals of the FORTH-83 International Standard.
- Uses the host operating system file structure (APPLE DOS 3.3 & CP/M 2.x).
- Built-in micro-assembler with numeric local labels.
- A full screen editor is provided which includes 16 x 64 format, can push & pop more than one line, user definable controls, upper/lower case keyboard entry, A COPY utility moves screens within & between lines, line stack, redefinable control keys, and search & replace commands.
- Includes all file primitives described in Kernigan and Plauger's Software Tools.
- The input and output streams are fully redirectable.
- The editor, assembler and screen copy utilities are provided as relocatable object modules. They are brought into the dictionary on demand and may be released with a single command.
- Many key nucleus commands are vectored. Error handling, number parsing, keyboard translation and so on can be redefined as needed by user programs. They are automatically returned to their previous definitions when the program is forgotten.
- The string-handling package is the finest and most complete available.
- A listing of the nucleus is provided as part of the documentation.
- The language implementation exactly matches the one described in <u>FORTH TOOLS</u>, by Anderson & Tracy. This 200 page tutorial and reference manual is included with MasterFORTH.
- Floating Point & HIRES options available.
- Available for APPLE II/II+/IIe & CP/M 2.x users.
- MasterFORTH \$100.00. FP & HIRES -\$40.00 each
- Publications
 - FORTH TOOLS \$20.00
 - 83 International Standard \$15.00
 - FORTH-83 Source Listing 6502, 8080, 8086 \$20.00 each.



A Professional Quality Z80/8080 Disassembler

REVAS Version 3

Uses either ZILOG or 8080 mnemonics Includes UNDOCUMENTED Z80 opcodes Handles both BYTE (DB) & WORD (DW) data Disassembles object code up to 64k long! Lets you insert COMMENTS in the disassembly!

A powerful command set gives you:

INTERACTIVE disassembly
Command Strings & Macros
On-line Help
Calculations in ANY Number Base!
Flexible file and I/O control
All the functions of REVAS V2.5

REVAS:

Is fully supported with low cost user updates
Runs in a Z80 CPU under CP/M*
Is normally supplied on SSSD 8" diskette
Revas V 3...\$90.00 Manual only...\$15.00

California Residents add 61/2% sales tax

REVASCO

6032 Chariton Ave., Los Angeles, CA. 90056 (213) 649-3575

*CP/M is a Trademark of Digital Resaerch, Inc.

Circle no. 54 on reader service card.

OPT-TECH SORT™

SORT/MERGE program for IBM-PC & XT

Now also sorts dBASE II files!

- Written in assembly language for high performance Example: 4,000 records of 128 bytes sorted to give key & pointer file in 30 seconds. COMPARE!
- Sort ascending or descending on up to nine fields
- Ten input files may be sorted or merged at one time
- Handles variable and fixed length records
- Supports all common data types
- Filesize limited only by your disk space
- Dynamically allocates memory and work files
- Output file can be full records, keys or pointers
- Can be run from keyboard or as a batch command
- Can be called as a subroutine to many languages
- Easy to use, includes on-line help feature
- Full documentation sized like your PC manuals
- \$99 —VISA, M/C, Check, Money Order, COD, or PO Quantity discounts and OEM licensing available

To order or to receive additional information write or call:

OPT-TECH DATA PROCESSING

P.O. Box 2167 Humble, Texas 77347 (713) 454-7428 Requires DOS, 64K and One Disk Drive

File Encryption (Listing continued, text begins on page 44) Listing Seven

```
3
                                                                     ;
ş
   SUBSF: The forward substitution function. Perform
        substitution on one 32-byte block of text by addition
;
        mod 2^256.
                 On call, DE -> text block, on page boundary
                                 in BLOCK common area,
3
                          HL -> key block, on page boundary in
;
                                 WORK common area.
                 On return, DE -> substituted text block
                 All registers are saved.
        CSEG
        MACLIB Z80
        PUBLIC
               SUBSF
SUBSF:
                 PSW
        PUSH
        PUSH
                 B
        PUSH
                 D
        PUSH
                 H
        MOV
                 A,E
                                  ;get DE pointing to last byte
        ADI
                 31
                                  ;...can do it this way since
        MOV
                 E, A
                                  ;...text is on page boundary
        MOV
                                  ;do the same for key block
                 A.L
        ADI
                 31
        MOV
                 L,A
        MVI
                 B, 32
                                  ; add 32 bytes
        ANA
                 A
                                  ;clear carry
FLOOP:
        LDAX
                 D
                                  ;get byte of text
        ADC
                 M
                                  ; add key to it
        STAX
                 D
                                  ; save sum in text area
                 H
        DCX
                                  ;decrement pointers
        DCX
                 D
        DJNZ
                 FLOOP
        POP
                 H
        POP
                 D
        POP
                 B
        POP
                 PSW
        RET
        END
                                                                End Listing Seven
```

Listing Eight

```
;
           The inverse substitution function. Perform
;
        substitution on one 32-byte block of text by
;
        subtraction mod 2^256.
:
                 On call, DE -> text block, on page boundary
                                 in BLOCK common area,
ş
                          HL -> key block, on page boundary in
÷
                                 WORK common area.
;
                 On return, DE -> substituted text block.
                 All registers are saved.
;
        CSEG
        MACLIB
                 Z80
        PUBLIC
                 SUBSG
SUBSG:
        PUSH
                 PSW
        PUSH
                 B
        PUSH
                 D
        PUSH
                 H
                                  ;get DE pointing to last byte
        MOV
                 A,E
                                  ... can do it this way since
        ADI
                 31
                                  ...text is on page boundary
        MOV
                 E.A
                                  ; do the same for key block
        MOV
                 A,L
                 31
         ADI
         MOV
                 L,A
                                  :subtract 32 bytes
                 B, 32
         MVI
                                  ;clear carry
         ANA
                 A
GLOOP:
                 D
                                   ; get byte of text
         LDAX
                                   ; subtract key from it
                 M
         SBB
                                   ;save sum in text area
                 D
         STAX
                                   :decrement pointers
         DCX
                 H
         DCX
                 GLOOP
         DJNZ
         POP
                 H
         POP
                 D
                 B
         POP
         POP
                 PSW
         RET
         END
```

End Listing Eight

;

File Encryption (Listing continued, text begins on page 44) Listing Nine

```
;
   PERMF:
           permute bit vector BLOCK by BITPERM:
;
           BLOCK <- BLOCK[BITPERM].
           On call, HL -> bit vector, BLOCK,
                     DE -> permutation table, BITPERM
           Assumes that BLOCK, BITPERM and WORK are on page
           boundary.
           Saves all registers.
        Macros
                         Ziloq
        _____
                         ====
;
        JRNZ nn
                         JR NZ, nn
        SETB n, M
                         SET n, (HL)
        BIT n,M
                         BIT n, (HL)
        MACLIB Z80
        CSEG
        PUBLIC
                 FERMF
        EXTRN
                RIPLO
                                  ;routine to clear workspace
PERMF:
        PUSH
                PSW
                                  ;save registers
        PUSH
                B
        PUSH
                D
        PUSH
                H
        CALL
                RIPLO
                                  ;clear partial result area
        LXI
                B, 0
                                  ; initialize indexes
                                  ; if Prth bit of BLOCK = 1 then
PRMF01:
                                  ;...Cth bit of WORK = 1
        LDAX
                D
                                  ;get BITPERM value
        RAR
                                  ; and convert to byte
        RAR
                                  :..index into BLOCK (divide by 8)
        RAR
        ANI
                 1FH
        PUSH
                H
                                 ; save BLOCKS's base
        ADD
                L
        MOV
                                 ;HL -> byte containing desired bit
        PUSH
                H
                                 ;save BLOCK pointer
        LXI
                H, EXBIT
                                 ;-> second byte of BIT instruction
        MVI
                A, 46H
                                 ;restore original BIT instruction
        MOV
                M. A
        LDAX
                D
                                 get BITPERM value again
        ANI
                07H
                                 :...and convert to bit index
        XRI
                07H
                                 reverses bit position in 0..7
        ADD
                A
                                 then shift to BIT's mask position
        ADD
                A
        ADD
                A
```

```
ORA
                 M
        MOV
                 M, A
        POP
                 H
                 O.M
        BIT
                 $-1
EXBIT
        EQU
                 PRMF20
                                   ;if BITPERM[i] bit is on, set
        JRNZ
                                   ;..Cth bit of WORK
PRMF10:
                                   :restore BLOCK base
        POP
                 H
                                   ;next BITPERM value
                 D
        INX
                 C
                                   :next bit of WORK
         INR
        DJNZ
                 PRMF01
                                   ;B is counting down from 255
        LXI
                 D. WORK
        XCHG
        LXI
                 B, 32
        LDIR
        POP
                 H
        POP
                 D
                 B
        POP
                 PSW
        POP
         RET
                                   ; ith bit of BLOCK is on; set on
                                   ;...Cth bit of WORK
PRMF20:
                                   ; find the jth byte of WORK
         MOV
                 A.C
         RAR
         RAR
         RAR
                 1FH
         ANI
         LXI
                 H, WORK
                 L
         ADD
         MOV
                 L,A
                                   ;save BLOCK pointer
         PUSH
                 H
                 H, EXSET
         LXI
                                   restore original SETB instruction
         MVI
                  A, OC6H
         MOV
                 M, A
                                   ; find the ith bit of the byte
         MOV
                  A,C
         ANI
                  07H
                  07H
         XRI
         ADD
                  A
         ADD
                  A
         ADD
                  A
         DRA
                  M
         MOV
                  M, A
         POP
                  H
                  O.M
         SETB
                  $-1
EXSET
         EQU
         JMP
                  PRMF10
                                    the temporary workspace; this
                  /WORK/
         COMMON
                                    ;routine uses only 32 bytes,
WORK
         DS
                  33
                                    ...but others need 33
         END
```

End Listing Nine

File Encryption (Listing continued, text begins on page 44)

Listing Ten

```
;
   PERMG:
           Inversely permute bit vector BLOCK by
;
           BITPERM: BLOCK <- BLOCK[1/BITPERM].
;
;
           On call, HL -> bit vector, BLOCK,
                     DE -> permutation table, BITPERM.
           Saves all registers.
        Macros
                         Ziloa
        =====
                         ____
        JRC nn
                         JR C, nn
        SETB n.M
                         SET n. (HL)
        CSEG
        MACLIB
                Z80
        PUBLIC
                 PERMG
        EXTRN
                 RIPLO
                                  ;routine to clear workspace
PERMG:
        PUSH
                 PSW
                                  ; save all registers
        PUSH
                 B
        PUSH
                 D
        PUSH
                 H
        PUSH
                 H
                                  ;save bit vector pointer
        CALL
                 RIPLO
                                  ;clear partial result area
        XRA
                 A
                                  ; get a zero
        STA
                 COUNT
                                  ;zero bit count
        MVI
                 B. 0
                                  ;zero index
        IVM
                 C, 32
                                  ;outer loop control
PRMG01:
        MVI
                 B, 8
                                  ;inner loop control
        MOV
                 A.M
                                  ;first byte of BLOCK
        PUSH
                 H
                                  ;save BLOCK pointer
        LXI
                 H, COUNT
                                  to count bits
PRMG02:
        ADD
                                  test bit for 1
        JRC
                 PRMG20
                                  ; if 1, set on BITPERM[i] bit
PRMG05:
                                  :.. of result
        INR
                 M
                                  ; count the bit
        DJNZ
                 PRMG02
                                  ; do for all bits
        POP
                 H
                                  ;restore BLOCK pointer
        INR
                 L
                                  ; next byte: BLOCK must be on page
        DCR
                 C
                                  ; adjust outer loop
        JNZ
                 PRMG01
                                  ; do for all bytes of BLOCK
        POP
                 H
                                  ;return pointer to start of BLOCK
        LXI
                 D. WORK
                                  replace input BLOCK with result
        XCHG
        MVI
                 C, 32
```

Assembler

New 5.0 Release

First Commercial Release January, 1983

The QueloTM portable 68000 assembler conforms to the Motorola resident assembler, publication M68KMASM[D4].

Quelo[™] 68000 Assembler Package Features:

Input file concatenation, include function, macros, global parameter substitution from command line, listing date-time parameter substitution from command line, listing date-time stamp, up to 31 character symbols, conditional assembly, structured programming directives, instruction optimization, 68010 instructions, relocation and linking, complex expression linking (all operators), DB-DW-DL directives for Z80 byte order data generation, object library utility, software configuration tracking, conditional linking, options for assembler and linker to write complete symbol table to a file, detailed symbol table listings, assembler symbol cross-reference, linker global symbol cross-reference, object library symbol cross-reference, superb linker load map, various HEX load formats produced by linker, error messages in English, extensive typeset manual with index, readily transported to any system with a C compiler and "UNIX like" system interface.

Ready to run in various disk formats for CP/M-80, -86, -68K, MS-DOS and PC-DOS

\$300 early bird price, good until June 30, 1984. \$595 after June 30.

Portable source version with detailed installation and testing instructions.

\$750 and license agreement.

processor type and specify kind of computer.

Model DE-4 U/V Products hold 8, 28 pin parts. High quality professional construction.

For more information or to order write or call Patrick Adams

QueloTM 2464 33rd Ave. W., Suite #173 Seattle, WA 98199 (206) 285-2528

COD, Visa, MasterCard.

TM of Intel Corp.

TM of Microsoft.

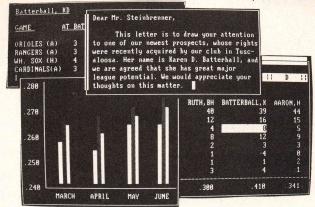
TM of Tandy Corp

Post Office Box 289

aveland, Mississippi 39576 [601]-467-8048

CP/M-80, CP/M-86, CP/M-66K TM DRI. UNIX TM Bell Labs. MS-DOS TM Microsoft. PC-DOS TM IBM.

SOFTWARE FUSION™—\$95



APX Core Executive allows you to:

- run up to 8 off-the-shelf programs for the IBM PC simultaneously in a multitasking window environment
- switch between programs with a single keystroke
- enjoy true concurrent processing
- move data between windows
- develop programs using our program interface
- build keystroke macros in a guick interactive fashion



CALL NOW

Application Executive Corp. 600 Broadway Suite 4C New York, N.Y. 10012

(212) 226-6347

5213H

X2816

48016

12816A

8741

8751

8742H

Circle no. 50 on reader service card

Circle no. 3 on reader service card.

2516

2532

2564 68766

68764 8755

2732A

27128

27C54



MODEL 7324 PAL PROGRAMMER Programs all series 20 & 24 PALS. Operates stand alone or via RS232.

File Encryption (Listing continued, text begins on page 44) Listing Ten

```
LDIR
                                   ; BLOCK <- BLOCK[1/BITPERM]
        POP
                 H
        POP
                 D
                 B
        POP
        POP
                 PSW
        RET
PRMG20:
        PUSH
                 PSW
                                   ; save current byte in A
        MVI
                 A, OC6H
                                   pristine 2nd byte of SET
        STA
                 EXSETG
        MOV
                 A.M
                                   ;get bit count, ith bit
        PUSH
                 D
                                   ;save BITPERM pointer
        MOV
                 E, A
                                   ; works because BITPERM is on page
        LDAX
                                   ;get BITPERM[i] byte
        RAR
        RAR
        RAR
        ANI
                 1FH
        PUSH
                 H
                                   ;save bit counter pointer
                 H, WORK
        LXI
                                   ; address partial result's ith byte
        MOV
                 L.A
        LDAX
                 D
                                   get BITPERM[i] byte again
        ANI
                 07H
                                   ;compute bit index into byte
        XRI
                 07H
                                   ;...count bits from left to right
        ADD
                                  then shift to SET's mask position
                 A
        ADD
                 A
        ADD
                 A
        PUSH
                 H
                                   ; save partial result ith pointer
        LXI
                 H, EXSETG
        ORA
                 M
        MOV
                 M.A
        POP
                                   restore partial result pointer
        SETB
                 0, M
EXSETG
        EQU
                 $-1
        POP
                 H
        POP
                 D
        POP
                 PSW
        JMP
                 PRMG05
COUNT
        DS
                 1
                                  ;holds bit count
        COMMON
                 /WORK/
WORK
        DS
                 33
                                  ; the temporary workspace; this
                                  ...routine uses only 32 bytes,
                                  ; but others need 33
        END
```

End Listing Ten

Listing Eleven

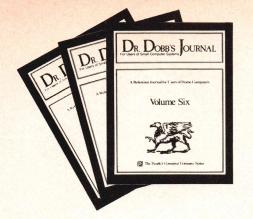
```
;
           Convert a permutation cycle to a permutation list
÷
        at the nth permutation.
5
        BITPERM <- CYCLE(N, RANDP, BITPERM)
;
÷
        On call: DE -> BITPERM (start of 256 byte area on
;
;
                         page boundary).
                   HL -> RANDP (256 bytes on page boundary).
3
                   A = N, 0 <= N <= 255.
        On return: BITPERM is established from RANDP and N.
;
        CSEG
        MACLIB
                 Z80
        PUBLIC
                 CYCLE
CYCLE:
                 PSW
        PUSH
        PUSH
                 B
        PUSH
                 D
        PUSH
                 H
                                  create BITPERM
                                  ; save N
                 OFFSET
        STA
                                  ;loop control and ith counter
        LXI
                 B, 0
                                  ;set DE to destination byte,
                                  ... DE+RANDP[C]
CYCLOOP:
                                  ; -> RANDP[C]
                 L,C
        MOV
                 A.M
        MOV
                                  ; DE -> BITPERM[RANDP[C]]
        MOV
                 E, A
                                  ;set HL to source byte
                 OFFSET
        LDA
        ADD
                 C
                                   : N+C
                                   ;HL -> RANDP[N+C]
        MOV
                 L,A
                 A,M
         MOV
                                   ;HL -> RANDP[RANDP[N+C]]
                 L,A
         MOV
                                   ;..get that byte and
                 A,M
         MOV
                                   ...put it in BITPERMIRANDP[C]]
                 D
         STAX
                 C
                                   ;next ith
         INR
                                   :do for all bytes
                 CYCLOOP
         DJNZ
         POP
                 H
         POP
                 D
         POP
                 B
                 PSW
         POP
         RET
OFFSET
         DS
                  1
         END
```

End Listing Eleven

(Listing Twelve begins on next page)

File Encryption (Listing continued, text begins on page 44) Listing Twelve

```
-
   UPTACK: Base conversion. ACCUM<-UPTACK(BASE, LENGTH, ARG)</pre>
-
     Convert digits represented by ARG to the base BASE.
     Output is in ACCUM (32 byte accumulator) in base 2.
     On Call: A = BASE: 1 < BASE < 256
              B = LENGTH of ARG in bytes: 0 < LENGTH < 256
              HL -> ARG: byte string representing digits in
                     base BASE: 0 <= byte < BASE.
     On return: HL -> base 2 representation of ARG, mod 2^256,
                 in 32 byte accumulator
        Macros
                         Zilog
        _____
                         ====
        RARR M
                         RR (HL)
        JRC nn
                         JR C, nn
        JRNC nn
                         JR NC, nn
        CSEG
        MACLIB Z80
               UPTACK
        PUBLIC
        EXTRN
                 RIPLO
                                 ;routine to clear accumulator
UPTACK:
        PUSH
                PSW
        PUSH
                 R
        PUSH
                D
        STA
                 BASE
                                  ;base used for shifting
        STA
                 BASER
                                  :..to refresh BASE
        SHLD
                 ARGP
                                  ; save argument pointer
        PUSH
                 B
                                  ;save loop control
        LXI
                 H, ACCUM
                                 ;clear accumulator
        LXI
                 D. ACCUM+1
        LXI
                 B, 31
        XRA
        MOV
                 M, A
        LDIR
        LHLD
                 ARGP
                                  ;restore pointer
        POP
                R
                                  ;restore loop control
        JMP
                UPTK02
UPTK01:
        CALL
                MULT32
                                 ; multiply accumulator by base
UPTK02:
        CALL
                ADD32
                                  ; add arg digit to accum
        INX
                                 ;next arg digit
        DJNZ
                UPTK01
        LXI
                H, ACCUM
                                 ;point to ACCUM on return
        POP
                D
```



Dr. Dobb's Journal

BOUND VOLUMES

Every Issue Available For Your Personal Reference.

Vol. 4 1979

This volume heralds a wider interest in telecommunications, in algorithms, and in faster, more powerful utilities and languages. Innovation is still present in every page, and more attention is paid to the best ways to use the processors which have proven longevity—primarily the 8080/Z80, 6502, and 6800. The subject matter is invaluable both as a learning tool and as a frequent source of reference.

Main subjects include: Programming Problems/Solutions, Pascal, Information Network Proposal, Floating Point Arithmetic, 8-bit to 16-bit Conversion, Pseudo-random Sequences, and Interfacing a Micro to a Mainframe—more than ever!

Vol. 5 1980

All the ground-breaking issues from 1980 in one volume! Systems software reached a new level with the advent of CP/M, chronicled herein by Gary Kildall and others (DDJ's all-CP/M issue sold out within weeks of publication). Software portability became a topic of greater import, and DDJ published Ron Cain's immediately famous Small-C compiler—reprinted here in full!

Contents include: the Evolution of CP/M, and CP/M-Flavored C Interpreter, Ron Cain's C Compiler for the 8080, Further with Tiny BASIC, a Syntax-Oriented Compiler Writing Language, CP/M-to-UCSD Pascal File Conversion, Run-time Library for the Small-C Compiler and, as always, even morel

Vol. 6 1981

Microcomputing was entering a technical maturity in 1981, while continuing to break new ground. This volume includes Dr. Dobb's first all-FORTH issue and the first Dr. Dobb's "Clinic" columns. There is continued coverage of CP/M and Small-C development, along with J.E. Hendrix's Small-VM and Santa Barbara Tiny BASIC for 6809—all here in one giant volume.

Articles include: Pidgin—A Systems Programming Language, The Conference Tree, Write Your Own Compiler with META-4, several exciting Z80 utilities, North Star tidbits, and morel

Vol. 1 1976

The material brought together in this volume chronicles the development in 1976 of Tiny BASIC as an alternative to the "finger blistering," front-panel, machine-language programming which was then the only way to do things. This is always pertinent for bit crunching and byte saving, language design theory, home-brew computer construction and the technical history of personal computing.

Topics include: Tiny BASIC, the (very) first word on CP/M, Speech Synthesis, Floating Point Routines, Timer Routines, Building an IMSAI, and more.

Vol. 2 1977

1977 found **DDJ** still on the forefront. These issues offer refinements of Tiny BASIC, plus then state-of-the-art utilities, the advent of PILOT for microcomputers and a great deal of material centering around the Intel 8080, including a complete operating system. Products just becoming available for reviews were the H-8, KIM-1, MITS BASIC, Poly Basic, and NIBL. Articles are about Lawrence Livermore Lab's BASIC, Alpha-Micro, String Handling, Cyphers, High Speed Interaction, I/O, Tiny Pilot & Turtle Graphics, many utilities, and even more.

Vol. 3 1978

The microcomputer industry entered its adolescence in 1978. This volume brings together the issues which began dealing with the 6502, with mass-market machines and languages to match. The authors began speaking more in terms of technique, rather than of specific implementations; because of this, they were able to continue laying the groundwork industry would follow. These articles relate very closely to what is generally available today.

Languages covered in depth were SAM76, Pilot, Pascal, and Lisp, in addition to RAM Testers, S-100 Bus Standard Proposal, Disassemblers, Editors, and much, much more.

ALL 6 for ONLY 3	125, a savirigs of o	VEI 1370!
Please charge my: Visa I enclose Check/money order	☐ MasterCard	☐ American Express
Card #	Expiration Date	
Signature		
Name	Address	

State

YESI Please send me the following Volumes of Dr. Dobb's Journal.

Mail to: Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303 Allow 6–9 weeks for delivery.

Vol. 1	x \$23.75 =
Vol. 2	x \$23.75 =
Vol. 3	x \$23.75 =
Vol. 4	x \$23.75 =
Vol. 5	x \$23.75 =
Vol. 6	x \$27.75 =
All 6	x \$125.00 =
Sub-total	\$

Postage & Handling
Must be included with order.
Please add \$1.25 per book in U.S.
(\$2.00 each outside U.S.)

TOTAL \$

File Encryption (Listing continued, text begins on page 44) Listing Twelve

```
POP
                  B
         POP
                  PSW
         RET
MULT32:
         PUSH
                  B
                                    ;save loop control
         PUSH
                  H
                                    ;save arg pointer
         CALL
                  RIPLO
                                    ;clear work area
         MVI
                  B, 8
                                    ;shifter loop control
MULTO1:
         LXI
                  H, BASE
         ANA
                  A
                                    ;clear carry
         RARR
                  M
                                    ; if least signif bit is on
         JRC
                  PARTIAL
                                    ;...sum partial product
MULTO2:
         PUSH
                  B
         MVI
                  B. 32
         LXI
                  H, ACCUM+31
         ANA
                  A
MULTO5:
                  M
         RALR
                                    ; double multiplicand by shift
         DCX
                  H
         DJNZ
                  MULT05
         POP
         DJNZ
                  MULT01
         LXI
                  D, ACCUM
                                    ;return product to ACCUM
                  H, WORK
B, 32
         LXI
         LXI
         LDIR
         LDA
                  BASER
                                    refresh base
         STA
                  BASE
         POP
                  H
         POP
                  R
         RET
PARTIAL:
                  D, WORK+31
         LXI
                                    ; add ACCUM to WORK
         LXI
                  H, ACCUM+31
         PUSH
                  B
         MVI
                  B, 32
         ANA
                  A
PARTO1:
         LDAX
                  D
         ADC
                  M
         STAX
                  D
         DCX
                  D
         DCX
                  H
        DJNZ
                  PARTO1
         POP
         JMP
                  MULT02
```

LXI D, ACCUM+31 LDAX D ADD M STAX JRC ADD05 RET ADDO5: PUSH B MVI B, 31 ADDO7: DCX D LDAX D ADI 1 STAX D **JRNC** ADD10 DJNZ ADD07 ADD10: POP B RET ARGP DS 2 BASE DS BASER DS 1 **ACCUM** 32 DS /WORK/ COMMON WORK DS 33

END

;save pointer to argument ;hold base for shifting

;hold original base for refresh

; accumulator

:common work area

End Listings

AVAILABLE

Back Issues

	1982	
No.64-Feb.	No.66-April	No.68-June
No.69 – July	No.70 – Aug.	No.71-Sept.
No.72 – Oct.	No.73 – Nov.	No.74 – Dec.
	1983	
No.75 – Jan.	No.76-Feb.	No.77 - March
No.78 – April	No.80-June	No.81 – July
No.82 – Aug.	No.83-Sept.	No.84-Oct.
No.85 - Nov.	No.86 – Dec.	
	1984	
No.87 – Jan.	No.88 – Feb.	No.89 – March
No 90 – April	No.91 – May	No.92-June
No.93 – July		

TO ORDER: send \$3.50 per issue to: Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303.

name			
address			

Circle no. 71 on reader service card.

state

3003

The BDS C Compiler . . .

"Performance: Excellent. Documentation: Excellent. Ease of Use: Excellent."

That's what *InfoWorld* said when we introduced the BDS C Compiler four years ago. Today, the updated **BDS Version 1.5** is even *better*.

First, the BDS is still the *fastest* CPM/80-C compiler available anywhere.

Next, the new revised user's guide comes complete with tutorials, hints, error messages and an easy-to-use index — the perfect manual for beginner or seasoned pro.

Plus, the following, all for one price: Upgraded file searching ability for all compiler/linkage system files. Enhanced file I/O mechanism that lets you manipulate files anywhere in your system. Support system for float and long via library functions. An interactive symbolic debugger. Dynamic overlays. Full source code for libraries and run-time package. Sample programs include utilities and games.

Don't waste another minute on a slow language processor. Order now.

Complete Package (two 8"SSDD disks, 181-page manual): \$150. Free shipping on prepaid orders inside USA. VISA/MC, C.O.D.'s, rush orders accepted. Call for information on other disk formats.

BDS C is designed for use with CP/M-80 operating systems, version 2.2. or higher, it is not currently available for CP/M-86 or MS-DOS.

Callagara

BD Software, Inc. P.O. Box 2368 Cambridge, MA 02238 (617) 576-3828

city

A Small-C Concordance Generator

he program presented in this article builds a concordance of a source text file. Included in other source programs, it can become a high-level software tool. The utility extends an internal sort capability, which incorporates the use of word lists within the sort input phase.

What Is a Concordance?

The root word, concord, implies agreement. If a scholar studies the collected works of an author and finds that an excerpt from one portion is nearly identical to others, then he or she says that they are in concordance. If the passages are examined and key phrases listed alphabetically, referring back to their source, then this secondary work is called a concordance. In effect, the efforts of the scholar to document the agreement of the various passages takes on the name of that agreement.

An early illustration of a concordance is a scholar's examination of the Bible; another, an analysis of the complete works of Shakespeare. Today, any cross-reference is a proper subset of a concordance, whose full meaning is now much broader.

How Is a Concordance Used?

A concordance is used when you want to index something back to a source work or collection of works. It isn't a data base program, and it doesn't modify files or cause you to do so; it extracts the words you decide to extract (or excludes those you desire left out) and produces an index to their source.

One Example

While not referring specifically to a

by John Staneff

John E. Staneff, Jr., Rt. 1, Box 801, Ellensburg, WA 98926. concordance generator and while reviewing another kind of software product entirely, Jerry Pournelle, in the April 1983 BYTE (see especially p. 348), described an excellent use for a concordance. In addition to being a columnist for BYTE, Pournelle is a well-known science fiction writer; he wanted to document his cast of characters, presumably for reuse in future writings. What Pournelle attempted to do was to extract his characters and to index them back to their source.

If he had had a concordance generator available, he could have located his characters by line number, page number, or paragraph (he even could have picked them up by chapter or book). If he wished, he could have used only those words that occurred in an inclusionary list, or he could have selected only those words that did not occur in

an exclusionary list. He could have manually verified and edited the resultant data structure as required. Used in this way, the concordance becomes a high-level index to the original work.

Another Example

If a book publisher produces a quality text, the publisher usually provides an index to it. Traditionally, this requires that a highly skilled proofreader actually read the manuscript and draw out words that ought to be indexed. These words then are sorted, and the sorted list is reviewed for each word's worthiness for inclusion in the index. Finally, the index is prepared.

This process is unlikely to provide an accurate or adequate index. (Have you turned to a book's index only to find that either it doesn't exist or it's incorrect?) If the typesetting is redone as

 Begin or set up the tree, which is re-executable during the program execution: cninit(list,type,dupl);

where

'list' is the character array containing the file name of the keyword list,

'type' is the integer type of the list, 0 indicates that lists aren't used,

1 is exclusionary, and 2 is inclusionary,

'dupl' is the integer indicator for duplicates,
0 is to keep all token-reference pairs, and
1 is to exclude duplicates

2. Push data onto the tree, requiring both the key word and the reference point: cnbuild(linenumb,text);

where

'linenumb' is the integer reference, and
'text' is the character array of text, each string
is terminated by a null,

- Order the tree, producing a sequence list of the sorted tree: cnorder();
- 4. Pop data off the tree, returning the key word and associated reference pointer: i = cnxtrct(token);

where

'i' is the integer reference pointer, and 'token' is the character array containing the key word returned (terminated by a null character).

Table

the work is revised, it is likely that the cross-references may not be corrected, or if corrected, not thoroughly.

If book indexing were automated, using inclusionary lists developed individually for several broad disciplines (such as medicine, computing, etc.), perhaps this ambiguity of publishing could be brought under control. Here a concordance generator could be used to great advantage.

Yet Another Example

How many students type their class notes? How many of them do so on their personal computer? And how many of them keep the files around after printing off the notes? If the notes were stored by lecture and indexed through a concordance generator, then the problem of how to study for an exam on a specific topic would be much simplified. (Just think of the advantage in an open book test!)

Going beyond the classroom, once the student becomes a doctor, lawyer, accountant, or computerist, wouldn't those indexed class notes be of interest in solving problems in professional practice? And why limit usage to scholastic topics? A recipe collection, treated as one large work, could surely be indexed as to all the various ways to fix hamburger.

A Concordance Is a Finding Tool

Think of the concordance routines as a high-level software tool. By themselves, they don't do much, but incorporated into a larger source program, they can perform significant tasks. The simplest way of using the routines is to treat them as an internal sort, keeping everything sent to them and returning it all at the end. (The only limit to this would be memory space.)

In such a simplistic view, each line of a poem might be sent separately, and the subsequent report would list only those lines occurring multiple times within the poem. Or perhaps a corporate branch office location table needs to be sorted. Because the routines are modular and includable, they are also reusable. The office phone book might be sorted by name, printed, and resorted by extension, all within the same program execution.

By increasing the level of sophistication of the calling module and electing to perform inclusionary or exclusionary checking, the routines can lead to much greater rewards. Perhaps the greatest reward comes when a series of steps culminates in a short list of particularly sensitive information drawn from a great amount of raw data. By cross-checking the index for the presence of two or more key words (or phrases) that refer to a single passage, one might quickly find answers to questions usually considered difficult to solve.

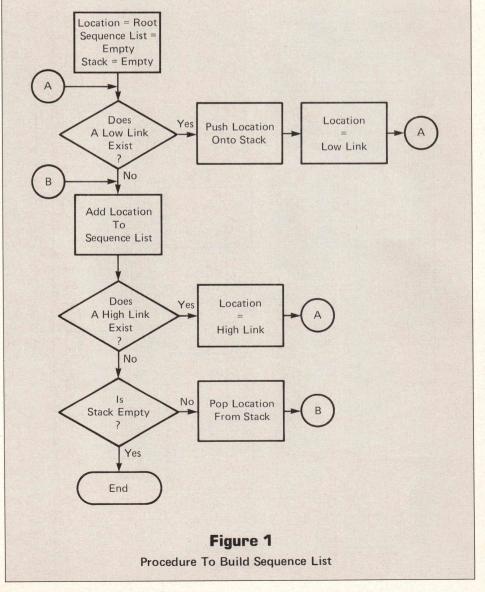
For instance, in what chapter of the Mabinogion does King Aurthur go hunting and is followed by Guenevere? If an index isn't handy, one might have to do a bit of reading! A more practical application might be to narrow down several symptoms to aid in illness diagnosis or to correlate law cases relating to a given set of circumstances. A pro-

gram that combines lists of a general nature (words like *a*, *the*, *an*, etc.) to condense a work and build an index, then passes that result against words of a specific nature (words like *compiler*, *ALGOL*, etc.) to refine the subset, and finally correlates those results by reference pointer is a powerful finding tool.

Using Lists To Select Words

The concordance routine, at the programmer's option, can use an external word list to govern its word storage activity. Two different kinds of lists may be used: an inclusionary list and an exclusionary list.

In fact, the difference between the two kinds of lists rests in how we humans regard them. Any word list can be used as either inclusionary or exclusionary at the programmer's option (or user's, if the choice is selected by an



externally supplied parameter).

The basic word list is simply constructed. Each entry is a word: an ASCII text string on a line by itself. The whole of the list is sorted alphabetically, and no duplicates are allowed. There are no special characters or hidden meanings and no trick start or end words; the list starts with the first word (on the first line) and ends with the end of file (no more lines in the list).

How the word list is used depends on the way it is described to the routines. If it is considered exclusionary, then only key words that do not appear in the list are accepted into the concordance routine's storage. If it is considered inclusionary, then only key words that appear in the list are accepted. In the end, only those words that have been accepted appear in the index. By definition, if a data file is passed against an inclusionary list to build an index and that index is passed against the same list used as an exclusionary list, a net loss of the entire data file will occur with no resultant index. However, it may be desirable to filter data in several steps, using exclusionary lists, and then to draw upon inclusionary lists to extract a desired index. (Conceivably, separate inclusionary lists can be used to build several indexes, which can then be merged to provide an interdisciplinary index.)

This technique is especially useful when the initial data file is quite large and has an abundance of noise words (such as articles, adverbs, and so on). For the final pass, the indexer may select from a number of specially prepared and maintained "stock" word lists, each word list corresponding to a

discipline's jargon or other criteria.

If multiple word lists are used to filter a source file, they may contain overlapping words. As long as the word lists are used consistently (i.e., both used as exclusionary lists), this will not present a problem. If they contain an overlap and aren't used consistently, the result is predictable: the overlapping words will *not* appear in the final result. One technique to "discover" overlapping words is to perform a simple cross-reference on one of the word lists while using the other word list as an inclusionary list. The result is a list of words present in both lists.

Certainly concordances may be prepared without using word lists, but employing the powerful utility that word lists can provide makes them invaluable tools.

The Programs

Presented here are three modules written in Small-C and developed on the early CP/M version as supplied by The Code Works. They are CNDEF.C, CNCORD.C, and CONCRD.C (Listings One, Two, and Three, pages 108, 108, and 109, respectively).

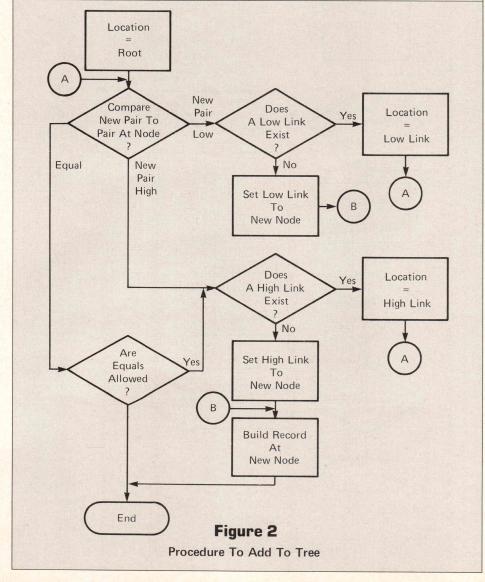
The CNDEF.C module sets up default variables required by the CNCORD.C module.

The CNCORD.C module builds the actual concordance, accepting tokens given it by the mainline program and stuffing them into internal storage. A reference number is also filed and can be used as line number, page number, paragraph number, chapter number, or entire work number.

The CONCRD.C module is a mainline program that makes use of the concordance modules. It is intended for demonstration only, producing a simplified concordance of a short text file. CONCRD.C also uses the modules as the heart of a cross-reference program, the major difference being that a concordance program retains the source lines while a cross-reference program retains references to already printed lines. The four calls to CNCORD.C are described in the table (page 88).

The Algorithm

Internally, the CNCORD.C routine is a binary tree sort. It interleaves source data acquisition with sorting and, upon



comand, dumps the tree's contents back to the caller. This technique is fairly quick. Once the data is read (and listed, if desired), the retrieval of sorted tokens proceeds rapidly. The apparent speed is the result of overlapping I/O delays with CPU activity in building the tree.

The attached flow charts (Figures 1 and 2, pages 89 and 90) describe the tree building and retrieval processes. Entries are added to the tree by finding an appropriate location in which to add the new token-reference pair and "linking" it in through the low and high tree links. Duplicate pairs are added only if the initialization call has specified that duplicates are allowed; when duplicates are added, they are considered higher than current entries.

Finding the correct location in which to add a pair is a process of testing each node for its relative value to the new pair, taking the low link if the new pair is lower and the high link otherwise. When there is no link address

to follow (because its value is zero), then that link address is made the new location, and the new pair is thereby linked.

Determining the order of the tree is an exercise in pushing and popping stacks. At each node, starting from the root (first in table), the node is tested for having a low link. If one is present, the node's address (table entry location) is pushed onto the stack, the low link is followed, and the process is repeated at the new node. If there isn't a low link, then the node's address is added to the sequence list. Next, if a high link is present, it is taken, with the process repeated (looking for a lower entry) from that node. If neither a high nor a low link is present, the stack is popped once, and the recovered address is added to the sequence list. The process continues from the point where it looks for a high link.

The process includes a sequence list, for the convenience of the output routine (cnxtrct(token)). When required,

the programmer can restart the output by resetting the list pointer (cnthrdpt) to zero.

A special routine included in the CNDEF.C listing (usrvfy(token)) is used to arbitrarily exclude any key word that may be identified without needing to be in a list. The version supplied will exclude all one- and two-letter words and any "word" that contains a nonalphabetic character. A sample version to accept all key words presented to it is also provided in the comments of the supplied routine.

(Listings begin on page 90)

Reader Ballot

Vote for your favorite feature/article. Circle Reader Service **No. 195**.

SMALL FOR IBM-PC

Small-C Compiler Version 2.1 for PC-DOS/MS-DOS Source Code included for Compiler & Library New 8086 optimizations Rich I/O & Standard Library



CBUG SOURCE LEVEL DEBUGGER FOR SMALL C

Break, Trace, and Change variables all on the source level
Source code included

Datalight
11557 8th Ave. N.E.

Seattle, Washington 98125

(206) 367-1803

ASM or MASM is required with compiler. Include disk size (160k/320k), and DOS version with order VISA & MasterCard accepted. Include card no. & expiration date. Washington state residents include 7.9% sales tax. IBM-PC & PC-DOS are trademarks of international Business Machines MS-DOS is a trademark of Microsoft Corporation.

541

You Read Dr. Dobb's Journal And You Don't Subscribe?

Save \$13 off newsstand prices for 2 yrs. Save \$5 for 1 yr.

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

Dr. Dobb's Journal 2464 Embarcadero Way Palo Alto, CA 94303

Yes! Sign me up for	2 yrs. \$471 yr. \$25
I enclose a check/i Charge my Visa, N Please bill me later	Master Card, American Express
NameAddress	
Credit Card	Exp. date
Acct No.	
Signature	300

Sample Run

B†type verse.txt Here with a loaf of bread beneath the bough, A flask of wine, a book of verse—and thou Beside me singing in the wilderness— And wilderness is paradise enow.

"How sweet is mortal sovranty!"—think some: Others—"How blest the paradise to come!" Ah, take the cash in hand and waive the rest: Oh, the brave music of a distant drum!

B†

The Input Text

B†concrd Input file name: verse.txt Exclusionary list name: xcl.lst

BENEATH

Here with a loaf of bread beneath the bough, BESIDE

Beside me singing in the wilderness—

BLEST

Others—"How blest the paradise to come!" BOOK

A flask of wine, a book of verse—and thou **BOUGH**

Here with a loaf of bread beneath the bough, BRAVE

Oh, the brave music of a distant drum! BREAD

Here with a loaf of bread beneath the bough, CASH

Ah, take the cash in hand and waive the rest; DISTANT

Oh, the brave music of a distant drum! DRUM

Oh, the brave music of a distant drum!

ENOW

And wilderness is paradise enow.

FLASK

A flask of wine, a book of verse—and thou

B†type xcl.lst SOME ALL TAKE THE AND COME THERE THRU FINI HAND TILL HERE WHEN **OTHERS** WITH OUR B† REST

The List of Words to Exclude

LOAF

Here with a loaf of bread beneath the bough, MORTAL

"How sweet is mortal sovranty!—think some: MUSIC

Oh, the brave music of a distant drum! PARADISE

And wilderness is paradise enow.

Others—"How blest the paradise to come!"

SINGING

Beside me singing in the wilderness—

SOVRANTY

"How sweet is mortal sovranty!"—think some: **SWEET**

"How sweet is mortal sovranty!"-think some:

THINK

"How sweet is mortal sovranty!"—think some:

THOU

A flask of wine, a book of verse—and thou VERSE

A flask of wine, a book of verse—and thou WAIVE

Ah, take the cash in hand and waive the rest:

WILDERNESS

Beside me singing in the wilderness— And wilderness is paradise enow.

WINE

A flask of wine, a book of verse—and thou B†

The Output from CONCRD.C

GGM — FORTH™ has HELP* for Z80¹ using CP/M²

GGM—**FORTH**, a complete software system for real-time measurement and control, runs on any Z80 computer under CP/M using an extended fig-FORTH vocabulary.

GGM—FORTH features:

- Open multiple CP/M files, in any combination of direct-access and sequential-access, fully compatible with all CP/M utilities
- Char. in/out uses CP/M console, lister, file, or port
- On-line HELP* provides instant access to definitions in the run-time GGM—FORTH dictionary
- HELP* file is easily extended to include user definitions using HELP* utility
- HELP* is available during full-screen editing

Complete system and manuals \$150.

Manuals only: \$20.

Introductory System: \$35.

GGM SYSTEMS, INC. 135 Summer Ave.. (617) 662-0550 Reading, MA 01867

¹Z80 is a trademark of Zilog, Inc. ²CP/M is a trademark of Digital Research, Inc.

Circle no. 25 on reader service card.

Thanks to YOU We're Growing with YOU and your Computer . . .



LEO ELECTRONICS, INC.
P.O. Box 11307
Torrance, CA. 90510-1307
Tel: 213/212-6133 800/421-9565
TLX: 291 985 LEO UR

We Offer . . . PRICE . . . QUALITY . . . PERSONAL SERVICE

64K UPGRADE

9 Bank	(IBM PC)	\$43.65	(150ns)
		\$41.85	(200ns)
4164	(150ns)	\$4.85 ea	
	(200ns)	\$4.65 ea	
8 Bank	(other PC)	\$38.80	(150ns)
		\$37.20	(200ns)
4164	(150ns)	\$4.85 ea	
	(200ns)	\$4.65 ea	
21	56K "Mothe	r-Savor" Hr	narado

8 - 256K - (150ns) \$400.00 6116P-3 - \$4.40 2732 - \$3.95 2716 - \$3.20 2764 - \$7.00

2716 - \$3.20 2764 - \$7.00 TMS-2716 - \$4.95 27128 - \$24.00

We accept checks, Visa, Mastercard or Purchase Orders from qualified firms and institutions. U.S. Funds only. Call for C.O.D. California residents add 6½% tax. Shipping is UPS. Add \$2.00 for ground and \$5.00 for air. All major manufacturers. All parts 100% guaranteed. Pricing subject to change without notice.

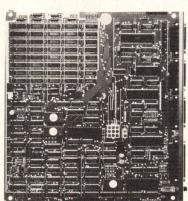
Circle no. 32 on reader service card.

Z80* SINGLE BOARD COMPUTER! 64K RAM — 80 x 24 VIDEO DISPLAY — FLOPPY DISK CONTROLLER RUNS CP/M* 2.2!

BRAND NEW
PC BOARDS
PC BOARDS!
WITH SCHEMATICS!

BOARD MEASURES

ALL ORDERS WILL BE PROCESSED ON A STRICT, FIRST COME, FIRST SERVED BASIS! ORDER EARLY!



\$29.95

(BLANK BOARD WITH DATA AND ROM'S.)

NEW PRICE

GROUP SPECIAL: BUY 6 FOR \$165!

USES EASY TO GET PARTS!

UNBELIEVABLE LOW PRICE!!! GIANT COMPUTER MANUFACTURER'S SURPLUS!

Recently Xerox Corp. changed designs on their popular 820* computer. These prime, new, 820-1 PC boards were declared as surplus and sold. Their loss is your gain! These boards are 4 layers for lower noise, are solder masked, and have a silk screened component legend. They are absolutely some of the best quality PC boards we have seen, and all have passed final vendor QC. Please note, however, these surplus boards were sold by Xerox to us on an AS IS basis and they will not warranty nor support this part.

We provide complete schematics, ROM'S, and parts lists. If you are an EXPERIENCED computer hacker, this board is for you! Remember, these are prime, unused PC boards! But since we have no control over the quality of parts used to populate the blank board, we must sell these boards as is, without warranty. You will have to do any debugging, if necessary, yourself!

*CP/M TM OF DIGITAL RESEARCH INC. (CALIF.) 820 TM OF XEROX CORP. Z80 TM OF ZILOG

WE ALSO CARRY LS, Z-80, EPROM'S, ETC. SEND FOR FREE CATALOG!

ADD \$2 PER PC BOARD FOR SHIPPING. (USA and Canada)

B. G. MICRO

P. O. Box 280298 Dallas, Texas 75228 (214) 271-5546

TERMS: Orders over \$50 add 85¢ insurance. No COD. Tex. Res. Add 6% Sales Tax. Subject to prior sale. Foreign orders: US funds only. We cannot ship to Mexico. Foreign countries other than Canada add \$6 per board shipping.

Concordance (Text begins on page 86) Listing One

```
1*
 * concordance global definitions and
 * variables.
 * Includes the usrvfy() routine, which should be
 * custom modified by user to further analyze tokens before
 * they are filed into the sort tree.
#define CNKEYSIZ
                         400
                                 /* number of keywords possible */
#define CNKEYSPC
                         2500
                                  /* amount of keyword space */
#define CNTKNLIM
                         1900
                                 /* number of tokens that can be sorted */
#define CNTKNLM
                         1899
                                 /* stack size, above minus one */
#define CNTKNSPC
                         12000
                                 /* size of sort tree char data */
#define CNLNSZ
                        80
                                 /* assume each line of 80 bytes */
#define CNDLSZ
                         30
                                 /* size of delimiter list */
int cnexcl[CNKEYSIZ]; /* pointer list */
char cnextxt[CNKEYSPC]; /* Text pointed to */
int cnexhi:
                        /* Search - high */
int cnexlo;
                         /* Search - low */
int cnlsfl;
                        /* Keyword file number */
int colsto:
                         /* List type */
int cosveqls;
                         /* 1 if save equals (token and reference), else 0 */
char cndelm[CNDLSZ]:
                                  /* Delimiter list */
char cntreedt[CNTKNSPC];
                                 /* token sort tree storage area */
int cntreerf[CNTKNLIM]; /* storage for references */
int cntreelc[CNTKNLIM]; /* pointers to text data for tree entry */
int cntreehi[CNTKNLIM]; /* pointer to next high in tree */
int cntreelo[CNTKNLIM]; /* pointer to next low in tree (or equal) */
int cntreesk[CNTKNLM]; /* stack of return-to pointers */
int cnthread[CNTKNLIM]; /* pointer list showing sequence of sort-tree */
int cnbegin;
                        /* sort-tree indicator */
int cnpos;
                        /* current location in sort-tree */
int cnsub;
                        /* for threading tree */
int cnsubr:
                       /* current sort-tree subscript */
int costart;
                        /* where sort-tree begins */
int costack;
                        /* number items in return-to stack */
                         /* current location in sequence of sort-tree */
int cothropt;
usrvfy(token)
char *token;
{
/*
* User verification routine, should be in user's application
 * module (or in cndef.c). In simplest form, is as follows:
*
        usrvfy(token)
 *
        char *token:
 *
                return(0);
```

```
* This simple version includes all commers.

* The version supplied here excludes (returns a 1) if the
  * token is less than 3 characters long, or if all characters
  * in the token aren't in range 'A' to 'Z'. Note that by
  * the time that tokens come here, they are all uppercase.

*/

int i;
char c;

i=0;
  while ((c=token[i])!=0) {
     if (c<'A') return(1);
     if (c>'Z') return(1);
     i++;
     }
  if (i<=2) return(1);  /* keep if three characters or more */
  return(0);
  }

/* end of cndef.c */

End Listing One</pre>
```

Listing Two

```
* Main code segment of the concordance routine.
 * Requires codef.c or equivalent to define the
 * global variables and sizes.
 * User may wish to rewrite the usrvfy() routine,
 * and to extend delimiter list in cninit().
cninit (file, 1sttype, savsw)
char *file;
int lsttype, savsw;
14
 * Initialize routine for concordance.
  "file" is filename of keyword file
 * "lsttype" is a switch relating to the keyword file
       0 = not used
        1 = acts as an exclusionary list
       2 = acts as an inclusionary list
  "savsw" is a switch relating to the need to retain duplicate references
       0 = keep all references
        1 = keep only one token per reference number
 * Returns O if ok
               1 if 1stfile not found (+ message on screen)
*/
int flag, c, loc;
        flag = 0;
                      /* assume all ok */
       cnlstp = 0;
                       /* default is list not used */
       if (lsttype==1) cnlstp=1; /* Exclusionary */
        if (lsttype==2) cnlstp=2;
                                     /* Inclusionary */
        if (cnlstp!=0) {
```

(continued on next page)

Concordance (Listing continued, text begins on page 86) **Listing Two**

```
cnlsfl=fopen(file, "r");
       if (cnlsfl==0) {
                puts("Can't open keyword file");
                putchar (CR);
                flaq=flaq+1;
        3
                /* default is to keep everything */
cnsveqls=0;
if (savsw==1) cnsveqls=1;
                              /* except unless requested otherwise *.
* The delimiter list is a string of those characters
 * which act as token separators, such as spaces and
 * commas.
 */
cndelm[00]=' ';
                        /* space */
cndelm[01]='
               7 :
                        /* tab */
cnde1m[02]=',';
                        /* comma */
cndelm[03]='.';
                        /* period */
cndelm[04]=';';
                       /* semi-colon */
                       /* colon */
cndelm[05]=':';
cnde1m[06]='!';
                       /* exclamation point */
cndelm[07]='?':
                       /* question mark */
cndelm[08]='-':
                        /* dash */
cnde1m[09]='%';
                        /* percent */
                        /* dollar sign */
cndelm[10]='$';
                        /* pound sign */
cndelm[11]='#';
cndelm[12]='*';
                        /* asterisk */
cndelm[13]='&';
                       /* ampersand */
                      /* open parenthesis */
/* close parenthesis */
cndelm[14]='(';
cndelm[15]=')';
cndelm[16]='@';
                      /* at sign */
                        /* equals sign */
cndelm[17]='=';
cndelm[18]='+';
                        /* plus sign */
                        /* tilde */
cndelm[19]='~';
                        /* end of list */
cnde1m[20]=0;
 * Note, the list might need to be extended, if so,
 * then also change the definition in cndef.c.
 */
/*
 * If the keyword list is available (file exists),
 * then read it into the stacks. Note that the routine
 * assumes that the input file is ordered. This would
 * require the user to pre-sort this file; not a bad
 * idea after any text-editing session.
 #/
cnexlo=0;
cnexhi=0;
if (cnlstp!=0) {
                         /* list should be used */
        if (flag==0) {
                 /* file is open... */
                 loc=0; /* main stacks subscript */
                cnexcl[++cnexhi]=loc; /* starting loc of text */
                while ((c=getc(cnlsfl))>0) {
                         if (c!=CR) {
```

```
if ((c)='a') & (c <='z')) c=c-32;
                                /* make table all uppercase */
                                cnextxt[loc++]=c;
                        else {
                                cnextxt[loc++]=0;
                                cnexcl[++cnexhi]=loc;
                        3
        /* at end of file, cnexhi is one higher than # keywords */
 * Initialize the sort-tree parameters too
 */
cnintree();
 * End of Initialization
          */
         return(flag);
 cnbuild(refr, text)
 int refr;
 char *text;
 1*
  * Routine to extract each token from the line "text", and to include
  * it in the sort-tree depending on the presence of the token in the
  * keyword list, and the exclude/include philosophy selected by the
  * The reference, "refr", is included along with the text as its
  * filed in the sort tree. This variable can be what ever the user
  * decides it to be, for instance it can be a line number or a page
  * number. In a more sophisticated approach, it can be used as a
  * paragraph number, chapter or verse number, or can be another type
  * of reference (ie: part of a linked list).
 char token[30];
 int 1, x, i, j, n, m;
 int p,q;
 char r;
        p=0;
        q=0;
        while ((r=text[p++])!=0) q++;
                        /* length of text/line including last null */
        1 = p-1;
        x=1;
         while (1) {
                ./* find and separate tokens */
                 i=cntokn(x,text);
                                      /* find next delimiter */
                 if (i==0) i=1; /* default to end of line if none found */
                 /* move "text" from "x" to "i" to "token" */
                 p=x-1;
                 q=0;
                 while (p < (i-1)) {
                         r=text[p++];
```

Concordance (Listing continued, text begins on page 86) Listing Two

```
token[q++]=r;
              token[q]=0;
                                    /* convert to uppercase */
              cnuprose(token);
               /*
                      */
                                      /* include/exclude test */
              n=cnfind(token);
              if (n==0) { /* not excluded */
                      m=usrvfy(token);
                      /* let application decide also */
                      if (m==0) {
                                      /* still ok */
                               cnstuff(refr, token);
                       3
              x=i+1:
              if (x>=1) break; /* continue to end of line */
                3
        3
cntokn(cnstart,text)
int costart;
char *text;
/*
 * Returns the positional location of the next delimiter character
* in the text string, from the starting position. Returns zero
 * if none found. All displacements are +1 relative to actual
 * position, including "cnstart" as supplied by caller.
 */
int i,j,k;
char c,s;
        while (i<(cnstart-1)) {
                /* find out if "cnstart" is past end of string */
                c=text[i++];
                if (c==0) return(0); /* Yep, too short */
        while ((c=text[i])!=0) {
                /* look at balance of string */
                while ((s=cndelm[j++])!=0) {
                        if (c==s) return(++i); /* delimiter found */
                        3
                i++;
                3
        return(0);
                       /* no delimiter found */
cnfind(token)
char *token;
{
 * Searches the keyword list to find a match to the token
 * presented.
```

Dr. Dobb's Journal NOW AT BIGGER SAVINGS!



If you take advantage of this	special o	offer yo	ou save o	over \$10	off news	stand prices
that's a 30% savings! Please charge my:Payment enclosed	Visa	M	asterCar _Bill me	d later	America	n Express
Card #					date	
Signature						
Signature						
Name						
Address						
City		S	tate	Zip		300
Offer good in USA only. Foreign This offer good until November A publication of M&T Publis	30, 1984.		. Please al	llow up to	six weeks	for first issue
Dr. Dobb's Journal				Reade	r Serv	ice Card
To obtain information about propriate number listed below. Use valid for 90 days from issue date.	bottom i	row to v	ote for b	est articl	e in issue.	
Name -			PI	none ()	
Address						
City/State/Zip						
1 2 3 4 5 6 7 8 9 10	11 12 1	3 14 15	16 17 1	8 19 20		
28 29 30 31 32 33 34 35 36 37 55 56 57 58 59 60 61 62 63 64						
82 83 84 85 86 87 88 89 90 91 109 110 111 112 113 114 115 116 117 118						
Articles: 190 191 192 193 194 195 196 197						
Comments:					se of Maga	
				2 □ Com 3 □ New 4 □ Boo 5 □ Pass	kstore ed on by fr	e riend/colleague
Dr. Dobb's Journal				Reade	r Serv	rice Card
To obtain information about propriate number listed below. Use valid for 90 days from issue date.	bottom	row to	vote for t	est artic	le in issue	cle the appro- This card is 1984, No. 94
Name			PI	hone ()	
Address						
City/State/Zip						
	11 12 13					24 25 26 27 51 52 53 54
28 29 30 31 32 33 34 35 36 37 55 56 57 58 59 60 61 62 63 64	38 39 40 65 66 67	7 68 69	70 71 72	73 74	75 76 77	78 79 80 81
82 83 84 85 86 87 88 89 90 91 109 110 111 112 113 114 115 116 117 118	92 93 94 119 120 121					05 106 107 108 132 133 134 135
Articles: 190 191 192 193 194 195 196 197						
Comments:					se of Maga	
Comments				1 □ Sub 2 □ Cor 3 □ Nev	oscription nputer Sto wsstand	
				4□Boo 5□Pas		riend/colleagu

6□Other_



BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMER

Dr. Dobb's Journal

P. O. Box 27809 San Diego, CA 92128 NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES



BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

Dr. Dobb's Journal

2464 EMBARCADERO WAY PALO ALTO, CA 94303 NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES





NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

Dr. Dobb's Journal

2464 EMBARCADERO WAY PALO ALTO, CA 94303



Changing Your Address?

Staple your label here.

To change your address, attach your address label from the cover of the magazine to this coupon and indicate your new address below.

Name

Address

Address

Apt. #

City State

Mail to: DDJ, 2464 Embarcadero Way, Palo Alto, CA 94303

The Cost Efficient EPROM Programmer!



DISPLAY □ Bright l" high display system □ Progress indicated during programming □ Error messages

KEYBOARD □ Full travel entry keys □ Auto repeat □ Illuminated function indicators

INTERFACE ☐ RS-232C for data transfer ☐ 110-19.2K baud ☐ X-on X-off control of serial data

FUNCTIONS □ Fast and standard programming algorithms

Single key commands

☐ Search finds data strings up to 256 bytes long ☐ Electronic signatures for easy data error I.D. ☐ "FF" skipping for max programming speed ☐ User sets memory boundaries ☐ 15 commands including move, edit, fill, search, etc. functions ☐ Extended mode reads

EPROM sets

GENERAL □ Stand alone operation, external terminal not needed for full command

set ☐ Total support ☐ 28 pin sockets ☐ Faulty EPROMS indicated at socket ☐ Programs 1 to 128K devices ☐ Built in diagnostics ☐ No calibration required ☐ No personality modules to buy ☐ Complete with 128K buffer ☐ Only

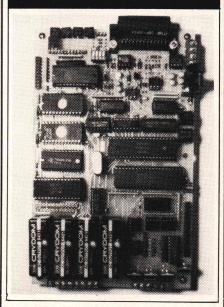
\$995.00 **COMPLETE**

Dealer inquiries welcome

SOUTHERN COMPUTER CORPORATION 3720 N. Stratford Rd., Atlanta, GA 30342, 404-231-5363

Introducing PROATROI®

The PROATROL[™] single board computer is both a low cost development system for programming in **FORTH**, and a powerful target system for a wide range of real time control applications.



The 5" x 8" stand alone board is based on the Rockwell single chip FORTH Microcomputer which contains a FORTH operating system in its internal ROM.

- RS 232 and 20 ma serial interface
- 4 sockets for plug in I/O modules (Opto 22, Crydom, etc.)
- 4 half bridge power drivers
- 8 optically isolated inputs
- 8 channel 8 bit ADC, ratiometric or referenced
- 4 potentiometers for level/speed settings
- 4 position dip switch
- 4 Schmitt trigger inputs
- 4 open collector outputs
- 8 expansion I/O lines
- Power fail interrupt/reset
- In circuit EPROM and EEPROM programming
- +5v regulator, DC/DC convertor
- Buss expansion connector
- Sockets for up to 14kb of memory

PROATROL™ has it all together!

Develop your program, configure your I/O options, debug your system and GO FORTH!

Only \$649.00

To order send remittance to:

PROA Corporation 4019 EDITH BLVD. NE BLDG. 2B

4019 EDITH BLVD. NE BLDG. 2B ALBUQUERQUE, NM 87107 For further information call

(505) 344-2106

Concordance (Listing continued, text begins on page 86) Listing Two

```
* Success or failure to match the token is translated to
* routine success or failure depending on "cnlstp".
* if colstp is 1, and token found, then exclude
                                       1, and token not found, then include
                                        2, and token found, then include
                                        2, and token not found, then exclude
*
                                     1 to exclude
   Returns
                                        O to include
* if "cnlstp" is 0, then always returns 0 (include)
*/
 int n,h,1,m,p,c,found;
 char test[30];
                       if (cnlstp==0) return(0);
                       found=0;
                                                               /* assume not found */
                       /* following routine is a binary search */
                       h=cnexhi; /* range limits */
                       l=cnexlo:
                       while (1) {
                                            n=(h+1)/2;
                                                                                   /* current pointer */
                                            m=cnexcl[n];
                                            /* move in the pointed-to keyword */
                                            p=0;
                                            while ((c=cnextxt[m++])!=0) test[p++]=c;
                                            test[p]=0;
                                            m=cncmpr(token,test); /* FDRTRAN-like compare */
                                             if (m==0) { /* found it */
                                                                  found=1;
                                                                  break;
                                             else if (m>0) 1=n;
                                             else h=n:
                                             if ((h-1) <= 1) {
                                                                                                      /* not in list */
                                                                 found=0;
                                                                  break;
                                                                  3
                                             3
                        /*
                           * Determine routine success based on following decision
                           * table:
                                        Keyword list not used
                           *
                                                                                                                                   Y
                           *
                                        Exclusionary list
                                        Inclusionary list
                           *
                                        Token found in list
                                        The class driving trade group of the class control 
                                        Ok, return(0)
                                                                                                                           X
                                                                                                                                        XX
                                        Fail, return(1)
                           */
                         if ((cnlstp==1) & (found==1)) return(1);
                         if ((cnlstp==2) & (found!=1)) return(1);
                        return(0);
```

```
cncmpr(a,b)
char *a, *b;
{
/*
 * Performs a FORTRAN-like compare.
 * Returns -1 if "a" is less than "b"
            O if "a" is equal to "b"
            1 if "a" is greater than "b"
 * Comparison is of strings, from left to right, continuing
 * so long as both are still equal, and then stopping if
 * either are 0 (NULL)
 */
int r,s,t,q;
char c,d;
        r=0;
        q=1;
        5=0;
        t=0;
        while (q==1) {
                c=a[s++];
                d=b[t++];
                 if (c<d) {
                         /* first is low */
                         r=-1;
                         q=0;
                else if (c>d) {
                         /* first is high */
                         r=1;
                         q=0;
                else if (c==0) {
                                        /* stop if end of string */
                         q=0;
        return(r);
cnuprese(text)
char *text;
/*
 * Routine translates all lower case letters to their
 * upper case equivalents.
 */
int i;
char c;
        i =0:
        while ((c=text[i])!=0) {
                 if ((c>='a') & (c<='z')) {
                                                  /* lower case found */
                         text[i]=c-32;
                                                  /* to upper */
                         3
                 i++;
                 3
        3
```

3

Concordance (Listing continued, text begins on page 86) Listing Two

```
cnintree()
/* Initialize the sort-tree */
        cnbegin=1;
        cnpos=0;
                        /* use 0 as null pointer, thus start at 1 */
        cnsub=1;
        cnstart=1;
>
cnstuff (refr, token)
int refr;
char *token;
/* Add token and reference to sort-tree */
int l,tst,c,next,link;
int i;
char entry[30];
         if (cnbegin==1) {
                                 /* turn off begin flag */
                 cnbegin=0;
                                        /* add this token as first in tree */
                 cntree(refr, token);
                 cnsub++;
         else {
                 cnsubr=cnstart;
                 tst=1;
                 while (tst==1) {
                         cngetree(entry,cntreelc[cnsubr]);
                         /* get entry at location of cnsubr */
                         c=cncmpr(token,entry); /* FORTRAN-like compare */
                         if ((c==0) & (cnsveqls==1)) {
                                  i=cntreerf[cnsubr];
                                  if (i==refr) return;
                         if (c<0) {
                                  next=cntreelo[cnsubr];
                                  link=2;
                         /* Place equal on high tree for proper output */
                         /* listing with respect to reference numbers */
                         if(c)=0) {
                                  next=cntreehi[cnsubr];
                                  link=1;
                         if (next==0) {
                                  tst=0;
                         else {
                                  cnsubr=next;
                 /* Thread in new entry when thread location found */
                 cntree(refr, token);
                 if (link==1) {
                         cntreehi[cnsubr]=cnsub;
                 else {
```

```
cntreelo[cnsubr]=cnsub;
                 cnsub++;
        3
cnorder ()
1*
 * This routine determines the retrieval sequence (order) of the
 * sort-tree. When the routine is complete, the array cnthread[]
 * is a pointer list to the sort-tree, in proper for a sequential
 * retrieval from the first entry.
 */
int tst;
int c:
int i:
char entry[30];
        cnthrdpt=1;
                         /* starting add-in entry for pointer list */
        cnstack=0:
        cnsubr=cnstart;
        tst=1;
        while (tst==1) {
                /* always look for a lower entry */
                c=cntreelo[cnsubr];
                if (c!=0) {
                         /* push stack when lower found */
                         cntreesk[++cnstack]=cnsubr;
                         cnsubr=c;
                else {
                         tst=cnpopstk();
        /* whole tree now ordered into pointer list */
        cnthrdpt=0;
                        /* prime retrieval pointer */
cnpopstk()
/* pop the top item in the tree cnstack */
char entry[30];
int c, t, tst;
        tst=1;
        t=1:
        while (tst==1) {
                /* no lower, print here */
                cngetree(entry,cntreelc[cnsubr]);
                cncrdout (cnsubr, entry);
                /* now look higher */
                c=cntreehi[cnsubr];
                if (c!=0) {
                         /* follows high link once, then lower */
                         cnsubr=cntreehi[cnsubr];
                         tst=0; /* end but not end of cnstack */
                else {
```

Concordance (Listing continued, text begins on page 86) Listing Two

```
/* pop stack when no higher found */
                         if (cnstack(=0) { /* are there any left in stack? */
                                 t=0:
                                 tst=0; /* end AND end of stack */
                         else {
                                 cnsubr=cntreesk[cnstack--]; /* pop again */
                         3
        return(t);
        }
cntree (refr, token)
int refr;
char *token;
/* Add line and reference to tree */
int t;
        t=cnpush(token,cnpos);
        cntreerf[cnsub]=refr;
        cntreelc[cnsub]=cnpos;
        cntreehi[cnsub]=0;
        cntreelo[cnsub]=0;
        cnpos=t;
cnpush (a,b)
char *a;
int b;
/* Moves contents of a into the position referenced by b in cntreedt */
int t,q,s;
char c;
         5=0;
         t=b:
         q=1;
         while (q==1) {
                 c=a[s++];
                 cntreedt[t++]=c;
                 if (c==0) q=0;
         return(t);
 cngetree (a,b)
 char *a;
 int b;
 /* moves the content of cntreedt[b] to a[?] until end of string */
 int t, q, s;
 char c;
         s=0;
         t=b;
         q=1;
         while (q==1) {
                  c=cntreedt[t++];
```

```
a[s++]=c;
                if (c==0) q=0;
                3
cocrdout (subr, entry)
int subr;
char *entry;
/*
 * Add current subscript pointer (cnsubr) to a-building ordered
 * pointer list. Assume that the variable cothropt is pointing
 * to the new location in the pointer list. (All new locations
 * are set to zero so that in the case of end of list, it is
 * properly identified.
 */
        cnthread[cnthrdpt++]=subr;
        cnthread[cnthrdpt]=0;
cnxtrct(token)
char *token;
{
1*
 * Function to return the sort-tree in order as determined by cnorder().
 * The global variable cnthrdpt, set to 0 by cnorder(), is assumed to
 * be the location of the previously listed token.
 * The variable "token" is set by the function when there
 * is data to return.
 * Function returns
                        O if end of list detected, no data values returned
                        else, returns the particular reference number
 */
int i;
int n;
        i=cnthread[++cnthrdpt];
        if (i!=0) {
                               /* get location of text for this subscript */
                n=cntreelc[i];
                i=cntreerf[i]; /* return i as the reference */
                cngetree(token,n);
                                        /* get text */
        return(i);
/* end of cncord.c */
                                                                   End Listing Two
```

Listing Three

```
/*
  * CONCRD.C
  *
  * This program presents a simplified concordance of an input file.
  *
  */
#define CR 13 /* carriage return */
char text[8000]; /* text storage area */
char filnm[50]; /* name of input file */
char tokn[80]; /* returned token */
```

(continued on next page)

Concordance (Listing continued, text begins on page Listing Three

```
ptokn[80];
                        /* prior returned token */
char
        line[80];
                        /* single text line */
char
int
        lines[200];
                        /* line pointers to text array */
                        /* number of lines stored */
        numlines;
int
                        /* variable associated to file */
int
        flvbl;
                        /* current byte on text[] */
int
        chrloc;
                /* init, load, sort, and list concordance */
main()
int lsttyp, savsw;
                        /* no lines yet stored */
        numlines=0:
                        /* no characters in text storage yet */
        chrloc=0;
                        /* open input file */
        opnfile();
                        /* Exclusionary List */
        lsttyp=1;
                        /* Save token only once per reference */
        savsw=1;
        puts("Exclusionary list name: "); gets(tokn);
                                        /* init concordance routine */
        cninit(tokn,lsttyp,savsw);
                        /* build concordance input arrays */
        loadtxt();
                        /* determine sequence */
        cnorder();
                        /* display concordance */
        prtcncrd();
        fclose(flvbl); /* close file */
                /* open input file */
opnfile()
        while (1) {
                puts("Input file name: "); gets(filnm);
                if ((flvbl=fopen(filnm, "r"))!=0) break; /* get file? */
                puts("Can't open file"); putchar(CR);
        3
                /* load text from input file into text array */
loadtxt()
int i;
                                                 /* while data */
        while ((i=getln(line,flvbl))>0) {
                                                 /* count lines */
                 lines[++numlines]=chrloc;
                                /* save in text array */
                mylnto();
                 cnbuild(numlines, line);
                                                 /* build concordance */
        3
                /* print concordance */
prtcncrd()
{
int i;
        ptokn[0]=0;
                        /* null token */
        while ((i=cnxtrct(tokn))!=0) { /* while tokens */
                                /* get line that token points to */
                mvlnfro(i);
                if ((i=cncmpr(tokn,ptokn))!=0) {
                                                         /* borrow routine */
                         puts(" "); putchar(CR);
                         puts(tokn); putchar(CR);
                         svtkn();
                                         /* save as new prior token */
                puts("
                            "); puts(line); putchar(CR); /* display line */
```

```
mvlnto()
                /* move contents of line to text at chrloc onward */
int i, j;
        j=0;
        while ((i=line[j++])!=0) text[chrloc++]=i;
        text[chrloc++]=0;
mvlnfro(i)
                /* move line by number from text to line */
int i;
{
int j,k,l;
        1=0;
        j=lines[i];
        while ((k=text[j++])!=0) line[1++]=k;
        line[1]=0;
svtkn()
                /* save current token in previous token */
{
int i,j,k;
        k=0; j=0;
        while ((i=tokn[j++])!=0) ptokn[k++]=i;
        ptokn[k]=0;
getln(input, file)
                         /* get line of input from file */
char *input;
int *file;
int c, i;
        i=0;
        while ((c=getc(file))>0) {
                input[i++]=c;
                if (c==CR) { input[i]=0; return(1); } /* (cr> ? */
        line[i]=0:
        return(c);
/* end of concrd.c */
#include a:cndef.c
#include a:cncord.c
```

End Listings

16-BIT SOFTWARE TOOLBOX

by Ray Duncan

Son of Floating-Point Benchmark

In case there are any newcomers out there, perhaps it is appropriate to supply some background to this month's column. We first published BASIC and PL-I versions of Bill Savage's floatingpoint speed and accuracy benchmark program in the September 1983 DDJ The subject seemed to catch the fancy of many readers, and versions of the benchmark were subsequently contributed and published for Pascal and Forth (November 1983), Fortran (January 1984), C (March 1984), Logo and LISP (June 1984), and 8086/8087 assembly language (July 1984). A reprise of the BASIC benchmark, a Modula-2 listing, T/Maker III. and a corrected Logo version accompany this month's column (Listings One-Four, pages 108-109).

A preliminary collection of benchmark results appeared in the March 1984 *DDJ* issue, and a greatly expanded table of results accompanies this month's column (page 111). We have timings for computers ranging from the mighty Cray X-MP to the humble Sinclair! We now have a very good representation of the programming languages, except for the little-known and arcane language known to its initiates as COBOL (apparently COBOL programmers read *Datamation* instead of *DDJ*).

As in the March table, the column FPP contains the type of hardware floating-point support that was used, if any was noted by the contributor. Most of the timings were accurate only to the nearest second (except for the very fast minicomputers and mainframes); they are displayed to three decimal places as an artifact of the dBASE II report program. The Error column was calculated as ABS-(2500.00000-A), converting to scientific notation and rounding the mantissa to the nearest integer. Obviously, the smallest errors correspond to the most accurate results.

There are some pretty interesting figures buried in the table. Digital Research Personal BASIC and Dimension 68000's BASIC lead the microcomputer field for grossness in absolute error, while IBM BASIC has the same distinction in the mainframe world. On the other hand, Digital Research Logo turned in an absolutely correct result; a classic example of one hand not knowing what the other hand is doing. Computer Innovations' new "Optimizing" C86 version 2 turned in a slower time than its previous, presumably "Non-Optimizing," C86 version 1.3. Morgan Computing's Professional BA-SIC interpreter for the IBM PC cranked out a timing and absolute error result competitive with the best compilers, while simply blowing the doors off every other known microcomputer interpreter.

Along with the timings and error results, I had the good fortune to receive many interesting and educational letters, some of which are excerpted below.

Benchmarking Spreadsheets

Thomas W. Moran of Bethesda, MD, writes: "I tried three spreadsheets (a perverse misapplication, I know!). Multiplan running on a Burroughs B20 (5 mHz 8086, I believe) took 24 minutes and gave an answer of perfect accuracy. T/Maker III, running on a 4 mHz Z80, took 31 minutes and gave 2500.00027709 as the answer. Running on a Nippon Univac system (5 mHz 8088, I think), T/Maker III gave the same answer but took only 27 minutes. Lotus 1-2-3 running on an IBM PC could only calculate up to 2048, not 2500 (at least without using a more complex program). I estimate its 2500 iteration time at about 8 minutes. It also gave a perfectly correct answer.

"Regarding accuracy: by adding an instruction to the T/Maker III program to request rounding to 9999,999999 after each iteration, I

forced T/Maker to give a perfectly correct answer. I suspect that both Multiplan and Lotus automatically round to integer or a few decimal places after each iteration, thus preventing the error from growing. I don't have convenient access to either, or I would have tried a starting value of, say, SORT (2), thus forcing non-integer calculations. A final subtraction of SORT(2) - 1.0 would then give results comparable to the other benchmark runs. Lotus was so much faster that I suspect it uses binary floating-point arithmetic. T/Maker uses decimal; I presume so does Multiplan. Does anyone know for sure?"

Benchmarking Modula-2

As everyone who reads Jerry Pournelle's *BYTE* column knows, Jerry's son Alex has proclaimed Modula-2 to be the language that will cure all the world's ills Real Soon Now. Chris Dunford writes:

"The attached listing is a Modula-2 version of the [Savage] benchmark, written for the M2M-PC compiler from the Modula Research Institute in Provo. Utah.

"The performance of the compiler in this particular test is far from impressive. Using single-precision reals in software (no 8087) on an IBM PC, the test took about 2 minutes and 38 seconds; the final value of A was 2262.9. A long time for such a bad result.

"The redeeming quality of the compiler is its price: \$40 plus a couple of bucks for shipping. The Institute is a nonprofit organization dedicated to spreading the gospel of Modula-2; the M2M compiler and its associated utilities were ported directly (in M-code) from the Lilith computer, Wirth's dedicated Modula machine. It appears that only the M-code interpreter and some low-level stuff were actually reworked for the PC.

"Although the system does not appear to be adequate for commercial

use, it's fine for learning the new language. The compiler fully supports Modula-2 as currently defined, and the M2M-PC package includes many of the library modules suggested by Wirth in the language specification, *Programming in Modula-2*. Just stay away from the floating-point support, and it's fine."

Benchmark Weak Points?

Kenneth M. Ferguson of Dallas, TX, writes: "When it became obvious that [this benchmark] was getting popular with the masses, I thought you might be interested in a similar benchmark that corrects some of the deficiencies inherent in the Savage benchmark. This new algorithm was suggested by Terry Peterson and appeared in the November issue of DTACK GROUND-ED, the newsletter from Digital Acoustics. Peterson's approach is as simple as the Savage benchmark but provides a more useful evaluation of the errors involved in floating-point calculations.

"There are two major problems with the Savage benchmark. It does not prevent errors of opposite sign from cancelling each other out.... A second drawback of the Savage algorithm is that the errors are weighted in favor of the larger test numbers. For example, assume a relative error of 0.01. Then, when n=1, the error contributed is 0.01. But when n=2500, the error is 25.0. Therefore, the larger test numbers contribute disproportionately to the final result.

"It is fairly simple to change the Savage algorithm into one where the errors are somewhat more interpretable. You simply collect a normalized squared error during the loop and then present the root mean square [RMS] error. For example:

loop n times
 normerror = (calculated/
 exact) - 1
 sumsquared = sumsquared +
 normerror * normerror
end loop
root mean square error = sqrt (sum-

"This algorithm solves both problems with the Savage benchmark. The cancellation of errors is prevented by squaring the deviation from the true



New Release

One user told us that, compared to other 8-bit C Compilers, Eco-C's "floating point screams". True. But, Release 3.0 has a number of improvements in other areas, too:

New optimizers with speed improvements of up to 50 percent over earlier releases!

New Compiler-time switches for greater flexibility.

A standard library with 120 pre-written functions.

Expanded error checking with over 100 possible error messages in English including multiple, non-fatal errors.

Improved, easy-to-read user's manual.

The Eco-C Compiler supports all data types (except bit-fields) and comes with MACRO 80 and the **C Programming Guide** for \$250.00. An optional, high-speed assembler and linker is available for an additional \$75.00. Eco-C requires a Z80 CPU, CP/M, and 56K of free memory. To order, call



5413 N. College Ave. • Indianapolis, IN 46220 (317) 255-6476



Eco-C (Ecosoft), CP/M (Digital Research), Z80 (Zilog), MACRO 80 (Microsoft)

Circle no. 21 on reader service card.

Introducing

WALTZ LISP T.M.

The one and only adult Lisp system for CP/M users.

Waltz Lisp is a very powerful and complete implementation of the Lisp programming language. It includes features previously available only in large Lisp systems. In fact, Waltz is substantially compatible with Franz (the Lisp running under Unix), and is similar to MacLisp. Do not be deceived by the low introductory price.

Waltz Lisp is a perfect language for Artificial Intelligence programming. It is also suitable for general applications. In fact, due to the ease of handling of textual data and random file access functions, it is often easier to write a utility program in Waltz Lisp than in any other programming language. Several general purpose utilities (including grep and diff) written entirely in Waltz Lisp are included with the interpreter.

Much faster than other microcomputer Lisps. • Long integers (up to 611 digits). Selectable radix. • True dynamic character strings. Full string operations including fast matching/extraction. • Random file access. • Binary files. • Standard CP/M devices. • Access to disk directories. • Functions of type lambda (expr), nlambda (fexpr), lexpr, macro. • Splicing and non-splicing character macros. • User control over all aspects of the interpreter. • Built-in prettyprinting and formatting facilities. • Complete set of error handling and debugging functions including user programmable processing of undefined function references. • Optional automatic loading of initialization file. • Powerful CP/M command line parsing. • Fast sorting/merging using user defined comparison predicates. • Full suite of mapping functions, iterators, etc. • Over 250 functions in total. • Extensive manual with hundreds of illustrative examples.

Waltz Lisp requires C/PM 2.0, Z80 and 48K RAM (more recommended). SS/SD 8" and most common 5" disk formats.



-INTERNATIONAL -

P. O. Box 7301 Charlottesville, VA 22906

Introductory Price....\$94.50

additional charges

\$10.00 conversion fee for 5" Diskettes \$3.00 C.O.D. charge

Call toll free 1-800-LIP-4000 Ask for Dept. #7 In Oregon and outside U.S.A. call 1-503-684-3000 Unix® Bell Laboratories. CP/M® Digital Research Corp.

Circle no. 46 on reader service card.

squared/n)

value, and each deviation is normalized so a true relative error is accumulated. Finally, the RMS error provides the average error for calculations over the range tested. Overall, it presents a more interpretable view of the errors in any given language's floating-point package.

"In the next issue of the newsletter from Digital Acoustics, another point is brought up. The arc tangent removes most of the error generated up to this step in the calculation. This is the result of having to map a number in the range from negative infinity to positive infinity into a range from -pi/2 to pi/ 2. Since most of the numbers in the benchmark have arc tangents that are very close to pi/2, any errors that have been picked up along the way are virtually removed. The error that is left comes from the error intrinsic to the arc tangent function The next step, the job of the tangent function, is to take an arc in the range -pi/2 to pi/ 2 and map it onto the range negative infinity to positive infinity. This is fraught with trouble, especially when all our test numbers are very nearly pi/2. The small error introduced by the arc tangent function becomes magnified by the tangent function. It turns out that the error introduced in this step is pi/2*N*error.

"To be redundant, let's write this another way. Let the test number be N, the correct result of the arc tangent be A, and the error introduced by the arc tangent function be 'err.' Then, the error after the tangent function will be

Tan(A+err)/N =relative error = err * pi/2

"The relative RMS error calculated by the Peterson algorithm is about 2267 times the relative RMS error plus the relative RMS errors of the rest of the functions. These latter errors will be almost entirely masked by the magnified arc tangent error. Therefore, dividing the relative RMS error by 2267 provides an estimate for the number of accurate bits in the floating-point package, unless the package has a particularly bad arc tangent function.

"In summary, the error in the arc tangent function dominates the errors accumulated by both the Savage and the Peterson algorithms by virtue of the magnification induced by the tangent function. The Peterson algorithm provides a way to estimate the accuracy of the floating-point package by making use of this dominance, a nice bonus."

DDJ

Reader Ballot

Vote for your favorite feature/article.

Circle Reader Service No. 196.

16-Bit (Text begins on page 106) Listing One

Bill Savage's floating-point speed and accuracy benchmark in BASIC.

```
100 ' time and general accuracy test program
110 defint i
120 iloop=2500
130 a=1
140 for i=1 to iloop-1
150 a=tan(atn(exp(log(sqr(a*a))))) +1
160 next i
170 print using "a=####.####";a
180 stop
```

End Listing One

Listing Two

Modula-2 version of the Savage benchmark, contributed by Chris Dunford.

```
(* DDJ Floating Point Test for M2M-PC Modula-2 Compiler *)
(*$R-,T-*) (* turn off run-time checks *)
MODULE fptest;
FROM MathLib@ IMPORT sqrt, exp, ln, arctan, sin, cos;
FROM RealInOut IMPORT WriteReal;
FROM Terminal IMPORT WriteString, WriteLn;
```

End Listing Two

Listing Three

T/Maker III code for Savage floating-point benchmark, contributed by Thomas W. Moran.

```
9999.999999
ex
ucl
     +*sqr
uc2 +log+exp
uc3
     +atn+tan
uc4+ 1
uc5
     pas
+
     1
ex
                                  2500 '+' lines
      . . .
ex
     9999,99999999
+
+
+
```

End Listing Three

Listing Four

Corrected Logo version of the Savage benchmark. This listing runs on "DR. LOGO"; the names of the transcendental functions may be slightly different for other implementations.

```
to savage
make "a l
repeat 2499 [make "a (tan arctan exp log sqrt :a * :a) + l ]
print [a = ] :a
end
```

End Listings

FLOATING POINT BENCHMARKS FOR DDJ

COMPUTER	MHZ	LANGUAGE	VERS	FPP	TIME	ERROR	SOURCE	COMMENTS
					(SEC)			
								0071101 TTME 0 00010 CEC
CRAY X-MP			X.11		0.000			ACTUAL TIME 0.00012 SEC
CRAY-1S		CFT FORTRAN	1.11		0.003		MARK SEAGER	
CDC 7600		FORTRAN	FTN5		0.069		MARK SEAGER	
CDC CYBER 170-875		BASIC			0.103		DAVID SACHS	
CDC CYBER 170-875		FORTRAN			0.144		DAVID SACHS	DOUBLE PREC.
CDC CYBER 170-875		FORTRAN			0.470		DAVID SACHS	SINGLE PREC.
HONEYWELL		MULTICS FORTRAN			0.500	4E+1		SINGLE PREC, TIME APPROX.
IBM 370		WATERLOO BASIC			0.570	2E-1	KARL CASPER	
DEC UAX 11/780		UMS FORTRAN-77			0.578	<1E-3	SHERMAN GROMME	SINGLE PREC
HP 1000-F		FORTRAN 4	2040		0.630	2E+2	COMPUSERVE	
IBM 3081		PL/I			0.660	2E-25	COMPUSERUE	
DEC 2060		TOPS-20 FORTRAN 4	V6		0.770	7E-1	P. O'KANE	SINGLE PREC.
DEC 20 KL-10		TOPS-20 FORTRAN	USA		0.830	1E+1	LEHMAN M. JONES	CPU TIME, SINGLE PREC.
HONEYWELL		MULTICS FORTRAN			1.000	1E-3	SHERMAN GROMME	DOUBLE PREC, TIME APPROX
DEC VAX 11-780		FORTRAN 77			1.000	7E-10	HORST SALZWEDEL	
UNIVAC 1100/81		FORTRAN	TFOR		1.020	2E-9	R. SCHLAIFER	DOUBLE PREC.
dill 1100 01			24J					
DEC UAX 11/780		UMS FORTRAN-77			1.054	<1E-3	SHERMAN GROMME	DOUBLE PRECISION
UNIVAC-1100/81		FORTRAN	FTH			2E-9		DOUBLE PREC.
OUITAIR TIONS OF		1000000	10R1					
DEC 2060		TOPS-20 BASIC+2			1.300	4E+1	P. O'KANE	SINGLE PREC.
DEC 2060		TOPS-20 FORTRAN 4	U6			1E+0	P. O'KANE	DOUBLE PREC.
IBM 370		IBM BASIC				2E+3	KARL CASPER	PRESUMED SINGLE PREC
DEC 20 KL-10		TOPS-20 FORTRAN	USA			8E-3	LEHMAN M. JONES	S CPU TIME, DOUBLE PREC.
HP A700		FORTRAN 4				3E-8	COMPUSERUE	
DEC VAX 11-780		C				7E-10	HORST SALZWEDE	
IBM PC (8088)	4.77	ASSEMBLER		8087		3E-10	CHRIS DUNFORD	
IBM PC (8088)	4.77	WL SYSTEMS FORTH		8087		2E-13	JOHN GOTWALS	
IPL 4441	1.11	PASCAL/US		0001		(1E-4	JEFF FURGAL	EQUIV. IBM 4343/2
8086	5.0	PL/I-86	1.01	8087		2E+1	BILL SAVAGE	MICROFLOAT LIBRARY
HP 9000 (68000)	3.0	FORTRAN (UNIX)	1.01	0001		2E-6		
HP 1000-F		FORTRAN FINA	2040) 5E-3		
8086	5.0	PL/I-86	1.01	8087		2E+1	COMPUSERVE	
	3.0	FORTRAN	1.01	0001) (1E-6	UNKNOWN	
HP 9000	4.77	MUP-FORTH		8087		(1E-3	C. SPRINGER	
IBM PC (8088)	1.11	RSX-11M FORTRAN		FIS		D 2E+2	JOHN TOSCANO	
DEC POP 11/44		PASCAL	2.1	113) 6E-10	COMPUSERUE	
DEC UAX 11/780		MS FORTRAN 77	3.1	8087		0 1E-9	HORST SALZWEDE	1 4/3/8/19
GRID COMPASS (8086)			3.1	8087		0 (1E-8	MICRO BUS. APP	
IBM PC (8088)	4.77	NS FORTRAN 77		8087		0 <1E-2	K. FERGUSON	
NEC APC (8086)	r 0			8087		0 1E-9	HORST SALZWEDE	1
IBM PC (8088)	5.0	ONX C MS FORTRAN 77	3.1	8087		0 1E-9	HORST SALZWEDE	
IBM PC (8088)	5.0	MICROSOFT PASCAL		8087		0 (1E-3	J. SPEISER	DBL PREC, SEATTLE 87.LIB
IBM PC (8088)	4.77	MICROSOFT FORTRAN	3.1	8087		0 (1E-3	J. SPEISER	DBL PREC, SEATTLE 87.LIB
IBM PC (8088)	1.11	RSX-11M FORTRAN	J.1	FIS		0 2E-1	JOHN TOSCANO	DOUBLE PREC
DEC PDP 11/44	4.77	CI C86	2 100	8087		0 2E-9	RICHARD LARSON	
IBM PC (8088)	1.11	RT-11 FORTRAN	2.100	FIS	7.50		J. SPEISER	A CONTRACTOR
DEC PDP 11/34 IBM PC (8088)	4.77	BASIC COMPILER	1 00	8087		0 1E-3	J. SPEISER	DBL PREC, SEATTLE 87.LIB
	1.11	BASIC CONFILER	1.00	1000		0 3E+2	COMPUSERVE	DOZ TREO, OLITTEE OT LEED
HP 1000F		DHOIC			0.00	0 05.6	OUT OUT OUT	

FLOATING POINT BENCHMARKS FOR DDJ

COMPUTER	MHZ	LANGUAGE	VERS	S FPP	TIME (SEC)	ERROR	SOURCE	COMMENTS
COMPUPRO 8088	5.0	DESMET C	2.2	0007		5E+0	R. WEITZMAN	DOUBLE DREE
COMPUPRO 8088	5.0	SUPERSOFT FORTRAN		8087		2E+2	R. WEITZMAN	DOUBLE PREC. SINGLE PREC.
8085	5.0	RMAC	1.01	8232		5E-3	BILL SAVAGE	MICROFLOAT LIBRARY
8085	5.0	PL/I-80	1 40	8232		5E-3	BILL SAVAGE	MICROFLOAT LIBRARY
8085	5.0	BASIC-80		8232	10.700		BILL SAVAGE	MICROFLOAT LIBRARY
NRT'L SEMIC. 16000		UNIX C	4.18	0232	12.000		DILL JHOHUL	DOUBLE PREC.
8085	5.0	FORTRAN-80		8232		5E-3	BILL SAVAGE	MICROFLOAT LIBRARY
COMPUPRO 8085	6.0	FORTRAN 80	3.3			2E+2	R. WEITZMAN	SINGLE PREC.
IBM PC (8088)		NC P-SYSTEM	C1F	8087	14.000		CHRIS DUNFORD	COMPILED TO NATIVE CODE
IBM PC (8088)	4.77	BASIC COMPILER		8087	14.200		J. SPEISER	
IBM PC (8088)	4.77	MORGAN PROF. BASIC		8087	15.200		RAY DUNCAN	DBL PREC, MICROWARE LIB
6502	4.0	UCSD P SYSTEM	II		15.500		STEVEN SPEARS	
8088	4.7	UCSD PASCAL				2E-7	COMPUSERVE	
LMC 16032	6.0	- C	1111	0001	18.000		R. SCHLAIFER	
APPLE II (6502)		MUP-FORTH		9511	18.500		C. SPRINGER	
LMC 16032	6	UNIX FORTRAN 77		3011	21.000		R. SCHAIFER	
DEC PDP 11/44		RSX-11M BASIC-PLUS-2				1E+2	JOHN TOSCANO	SINGLE PREC
DEC MINC 11/23		FORTRAN		FPU	22.000		BILL SAVAGE	JINULE FREE
DEC LSI 11/23		FORTRAN			25.000		K. FERGUSON	SINGLE PREC.
DEC PDP 11/44		RSX-11M BASIC			27.200		JOHN TOSCANO	JINULL FREL.
HEATH H-89 (Z-80)	4.0	FORTRAN-80		9511	31.000		JOHN TOSCANO	
DEC PDP 11/44		RSX-11M FORTRAN		3011	34.600		JOHN TOSCANO	
DEC MINC 11/23		MINC BASIC	2.0	FPU	38.000		BILL SAUAGE	
HP 9186		HP BASIC			44.000		COMPUSERVE	
HP 9836 (68000)		BASIC			44.290		BILL SAVAGE	
IBM PC (8088)	4.77	BASIC COMPILER	1 00		48.000		J. SPEISER	SINGLE PREC
SAGE (68000)	8.0	P-SYSTEM PASCAL	4.12		54.000		JOE BARNHART	SINGLE PRECISION
DEC LSI-11/23		C (UNIX)	U?		56.000		COMPUSERUE	SIMULE PRECISION
GRID COMPASS (8086)		GRIDBASIC		8087		1E-9	HORST SALZWEDEL	
SAGE (68000)	8.0	P-SYSTEM FORTRAN		0001		3E+2	JOE BARNHART	SINGLE PRECISION
SAGE (68000)	8.0	P-SYSTEM BASIC	4.12			3E+2	JOE BARNHART	SINGLE PRECISION
DEC LSI 11/23		FORTRAN			59.000		K. FERGUSON	DOUBLE PREC.
NEC APC (8086)		CI C86		8087	65.000		K. FERGUSON	BOODLE TREE.
DEC LSI-11/23		PASCAL-2	2.1A		66.000		COMPUSERVE	
WANG PC (8086)	8.0		1.0.2		68.000		RAY DUNCAN	
IBM PC (8088)	4.77		1.07		70.000		HUGH KAWABATA	SINGLE PREC
DIMENSION (68000)	0.8	BASIC			76.000		JOE BARNHART	40 X 24 SCREEN
SUN WORKSTN (68000)		UNIX CC COMPILER			85.000		JOHN TOSCANO	TO IT 21 SCREEN
IBM PC (8088)	4.77		1.330	8087	85.000		J. SPEISER	DOUBLE PREC
8086	5.0	BRSIC-86	5.20		92.200		BILL SAVAGE	DOGGEE TREE
DEC PDP 11/44		RSX11M FORTRAN			103.000		JOHN TOSCANO	DOUBLE PREC
DIMENSION (68000)	8.0	BASIC			103.000		JOE BARNHART	80 X 24 SCREEN
DEC PDP 11/44		RSX-11M BASIC-PLUS-2			113.600		JOHN TOSCANO	DOUBLE PREC
68000	8.0		IU.12		115.000		COMPUSERUE	
IBM PC (8088)	4.77	IBM APL			117.000 1		J. SPEISER.	
HP 9825T		HPL			117.000		WAYNE BORGLUM	
6809	2.0	PASCAL			119.000 8			MICROWARE SYSTEMS CORP
8088	8.0	DIGITAL RESEARCH C			119.000		GUY SCHARF	
HP 1000E		FORTRAN 4	2026		121.000 3	8E-8	COMPUSERUE	

Table I

(Continued on next page)

FLOATING POINT BENCHMARKS FOR DOJ

COMPUTER	MHZ	LANGUAGE	VERS	FPP	TIME (SEC)		SOURCE	COMMENTS
8085	5.0	FORTRAN-80	3 40		140.800	2E+2	BILL SAVAGE	
8085	5.0	BASIC-80 COMPILER			140.800		BILL SAVAGE	
HP 9835B	3.0	BASIC	3.20		140.800		J. SPEISER	
ZENITH Z-100 (8088)		ZBASIC			142.730		BILL SAVAGE	
6809	2.0	BASICO9			149.000		ANDY BALL	MICROWARE SYSTEMS CORP
8088	8.0	SSS FORTRAN	1 04		154.000		COMPUSERUE	
IBM PC (8088)	4.77	BASIC INTERPRETER			157.000		J. SPEISER	SINGLE PREC
IBM PC (8088)	4.77	M2M-PC MODULA 2			158.000		CHRIS DUNFORD	
IBM PC (8088)	4.77	BASIC COMPILER	1.00		170.000		J. SPEISER	DOUBLE PREC
8086	8.0	MS PASCAL	?		171.000		COMPUSERUE	
UAX 11/750	0.0	FRANZ LISP			172.000		MICHAL YOUNG	4 USERS, TIME APPROX.
8085	5.0	BASIC-80	5.20		174.900		BILL SAUAGE	
8086	5.0	PL/I-86	1.01		179.600		BILL SAUAGE	
Z-80	4.0	BRSIC-80	5.30		184.000		COMPUSERUE	
HEATH H-89 (Z-80)	4.0	MBRSIC COMPILER			189.700		JOHN TOSCANO	
HEURIKON (68000)	8.0	UNIX SYSTEM III C	3.6		199.300		BOB HALLORAN	DOUBLE PREC.
HEATH H-89 (Z-80)	4.0	FORTRAN-80			203.000		JOHN TOSCANO	
TANDY 16B (68000)	1.0	XENIX CC COMPILER			211.000		JOHN TOSCANO	
IBM PC (8088)	4.77	SUPERSOFT FORTRAN	1.07		211.000		HUGH KAWABATA	DOUBLE PREC
68000	6.0	C (TRS-XENIX)	1.01		212.000		COMPUSERUE	
IBW bc (8088).	4.77	DRI PERSONAL BASIC	1.0		217.000		RAY DUNCAN	SINGLE PREC
HEATH H-89 (Z-80)	4.0	MBASIC	4.8		229.800		JOHN TOSCANO	
	8.0	P-SYSTEM PASCAL	4.12		234.000		JOE BARNHART	DOUBLE PRECISION
SAGE (68000)	0.0	P-SYSTEM FORTRAN	4.12		236.000		JOE BARNHART	DOUBLE PRECISION
SAGE (68000)		MBASIC	5.22		236.740		BILL SAVAGE	
ZENITH Z-100 (Z-80) 8085	6.0	C/80	3.0		238.00		COMPUSERVE	
SAGE (68000)	8.0	P-SYSTEM BASIC	4.12		238.000		JOE BARNHART	DOUBLE PRECISION
DEC PDP 8	0.0	OS/8 FORTRAN IV	1.12		240.00		SHERMAN GROMME	
8085	5.0	PL/I-80	1.30		254.40		BILL SAVAGE	
IBM PC (8088)	4.77		1.0		255.00		RAY DUNCAN	DOUBLE PREC
68000	6.0	BASIC/S-16	1.7		266.00		COMPUSERVE	
UECT.GR. 4 (Z-80)	0.0	BASIC INTERPRETER	5.213			0 2E+2		DOUBLE PRECISION
HP 85		HPL HPL	0.210	旗	282.00		WAYNE BORGLUM	基面提供 医氯色
HP 86-A		BASIC				0 1E-3	MARK BAILEY	NATIVE MODE
8088	8.0	COMP. INNOV. C86	1 33		288.00		COMPUSERUE	
HEATH H-89 (Z-80)	2.0	BHBASIC				0 2E+1	PEARSON/CAIRO	
BBC ACORN (6502)	2.0	BBC BASIC	U2		311.28		P. O'KANE	SINGLE PREC.
HP-75 CALCULATOR	2.0	BASIC				0 6E-4	COMPUSERUE	
Z-80	4.0	BASIC-E	2.2			0 5E+1	COMPUSERUE	
NEC 8201	1.0	MS BRSIC				0 2E+2	HORST SALZWEDE	
HEATH H-89 (Z-80)	4.0	C/80	3.0			0 3E+2	JOHN TOSCANO	
Z-80	4.0	PL/I-80	1.3			10 9E+2	COMPUSERUE	
OSBORNE I (Z-80)		C/80				0 2E+2	BOB BRIGGS	
TANDY I (Z-80)		FORTRAN-80				0 2E+2	KARL CASPER	SINGLE PREC
APPLE II (6502)	1.75	APPLESOFT				0 (1E-3	C. SPRINGER	
APPLE II & 8088	1.75	APPLESOFT BASIC				00 (1E-3	R. S. SCHLAIFE	R
Z-80	4.0	NEUADA FORTRAN	2.2			0 8E+1		
VIC-20 (6502)	1.0	BASIC	2.0			10 9E-5		
IBM PC	4.77	LOTUS 1-2-3			480.00	0 (1E-3	THOMAS MORAN	2048 ITER. TIME APPROX.

Table I

(Continued on page 114)

WHAT TO C

Deluxe CP/M -- ISIS Package Includes: ICX file transfer program for 8" ISIS disks; ISE emulator allows ISIS software to run under CP/M. \$89 each or \$175 both

EZ-ZAP General purpose EPROM programming software utility. Uses simple parallel port interface to EPROM or adapts to other hardware. Includes schematic.

C-PACK A disk full of useful CP/M utilities written in C. PEP, DEL, BACKUP, DPATCH, ECHO, CHX, PAUSE, and more! \$19

Programs include complete source code CP/M and MP/M are registered trademarks of Digital Research, Inc. ISIS-II and MULTIBUS are registered trademarks of Intel Corp.

Western Wares

303 • 327 - 4898

Box C • Norwood, CO 81423

Circle no. 65 on reader service card.

SAL/80® and SAL/86™

do for assembly language what RATFOR does for FORTRAN but emits

OPTIMALLY DENSE code.

SAL/8X includes console I/O primitives which trivialize the task of writing complex interactive user interfaces. Improves programmer productivity by a factor of two and program maintainability by an order of magnitude.

Extensively documented, available for all CP/M compatible disk formats. SAL/80 version 2.1, \$59.00, requires 64K and MAC or RMAC.

CALIFORNIA RESIDENTS ADD 6% SALES TAX.

PROTOOLS® "Software Tools for the Professional"

24225 Summerhill Avenue Los Altos, CA 94022 (415) 948-8007

Circle no. 49 on reader service card.



CP/M KEYBOARD EXPANDER

POWERFUL NEW UPGRADE! still only \$100



microSystems

CP/M tm Digital Research, Inc.

Dealer and OEM inquiries welcomed

The National Software Show-Booth No. 2404

Circle no. 48 on reader service card

ATTENTION SOFTWARE AUTHORS

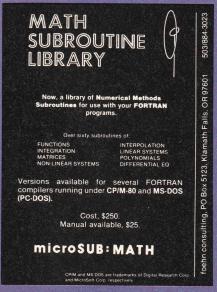
Our established literary agency is seeking to represent talented software authors. If you've written "The Great American Program" but are unsure how to market it or to whom or for how much, call us

Whether it's recreational, educational or business, we'll make sure that your creation gets the audience it deserves -Top Executives at the leading software

Our clients receive better treatment and earn more royalties because we negotiate the best possible deals. Give us a chance to go to bat for you. For further information on the benefits of representation please contact:

> THE ROBERT JACOB AGENCY (805) 492-3597 1642 Eveningside, Suite 110 Thousand Oaks, CA 91362

Circle no. 55 on reader service card.



Circle no. 24 on reader service card.

PROMPT DELIVERY!!!

DYN	IMAI	C RAM	
256K	150	ns	\$48.99
64K	200	ns	5.44
64K	150	ns	5.37
64K	120	ns	6.80
16K	200	ns	1.21
	EPR	OM	
27256	250	ns	Call
27128	250	ns	\$26.40
2764	200	ns	10.65
2732	450	ns	5.40
2716	450	ns	3.60
ST	ATIC	RAM	
5565P-15	150	ns	\$39.97
6264LP-15	150	ns	39.97
6116P-3	150	ns	6.36

Factory New, Prime Parts MICROPROCESSORS UNLIMITED 24,000 South Peoria Ave. (918) 267-4961 BEGGS, OK. 74421

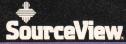
Prices shown above are for May 17, 1984
Prices shown above are for May 17, 1984
Prices shown above are for May 17, 1984
Prices shown and insurance extra. Cash
own. Small order received by 6 PM CST can usually be delivered to
forning, via Pederal Express Standard Air via 35,991

Circle no. 38 on reader service card.

energies, join The SourceView Corporation's SOFT WARE PUBLISHING PROGRAM. Get all of your R & [eeds at our distributor cost or loan, plus a lucrative

SourceView is small enough to give you personalized service, we register your software in ISBN publishing system, offer a nonexclusive marketing agreement system, ofter a nonexclusive marketing agreement, super packaging, specialized services. We seek: com-pilers, cross-assemblers, utilities, new DOS, DBMS, integrated information processors, graphics systems, educ., scien., bus., stat., eng. applications. We offer full development & documentation. Don't hesitate, con-tact SourceView immediately!

Michael L. Dean, VP Research & Development The SourceView Corporation Post Office Box 578, Concord, CA 94522 (415) 680-0202



Circle no. 62 on reader service card.

68000 Cross Assembler Motorola VERSAdos + Compatible

Assembler, Linker, Object and Macro Librarian. Absolute and Relocatable Code, Macros, Includes, and Conditional Assembly. Structured Programming. No limit on source file size.

> Unix (C) Compatible Source \$700

CP/M-80* \$200

PC/DOS† \$250

CP/M-86* \$250

Manual: \$20 (refundable)



1329 Gregory Wilmette, IL 60091 (312) 251-5310 after 5 p.m.

*Digital Besearch trademark + HBM trademark + Motorola trademark

Circle no. 23 on reader service card.

Advertisers! Get Ready For December Dr. Dobb's Journal

special UNIX issue

Space Reservation Deadline OCTOBER 12, '84 Materials Deadline **OCTOBER 19. '84**

> Contact: Walter Andrzejewski Alice Hinton (415) 424-0600

Dr. Dobb's Journal

2464 Embarcadero Way Palo Alto, CA 94303

Circle no. 70 on reader service card.

FLOATING POINT BENCHMARKS FOR DDJ

COMPUTER	MHZ	LANGUAGE	UERS	FPP	TIME (SEC)	ERROR	SOURCE	COMMENTS
TANDY I (Z-80)		MOLINERY PASCAL	5.1		485.000	1E+2	KARL CASPER	
TANDY I (Z-80)		BASIC			512.000		KARL CASPER	维持整理。
COMMODORE 64 (6510)		BASIC			514.000		TERRY THOMAS	
COMMODORE		PETSPEED	2.6		515.400		DTACK GROUNDED	
NEC APC (8086)		CI C86			524.000		K. FERGUSON	
KAYPRO II (Z-80)	2.5	TURBO PASCAL			538.300		MICHAL YOUNG	
Z-80	4.0	ZBAS	1.4		540.000	2E+2	COMPUSERUE	
IBM PC (8088)	4.77	TURBO PASCAL	1.00		544.000		JEFF FURGAL	TAN FUNCTION SUPPLIED
TANDY CC (6809)		FORTH			560.000		GARY BERGSTROM	
TANDY CC (6809)		BASIC RS-CC	1.0		585.000		GARY BERGSTROM	
HEATH H-89 (Z-80)	2.0	MICROSOFT BASIC	1. <u>1</u>		620.000		PEARSON/CAIRO	
APPLE MACINTOSH		MS BASIC	1.0		643.000		HORST SALZWEDEL	DOUBLE PREC.
ZENITH ZW110 8088	5.0	CI C86			655.000		A. F. HERBST	
IBM PC (8088)	4.77	CI C86	1.330		695.000		J. SPEISER	DOUBLE PREC
IBM PC (8088)	4.77	DR. L060	1.0		750.000		RAY DUNCAN	
IBM PC (8088)	4.77	CI C86	2.100		764.670		RICHARO LARSON	DOUBLE PREC.
TANDY 168 (68000)		XENIX MBASIC	2		773.600		JOHN TOSCANO	
IBM PC (8088)	4.77	CI OPTIMIZING C86	2.050		774.000		JEFF FURGAL	DOUBLE PREC.
GRID COMPASS (8086)	1.11	DESMET C	2.2		796.000		HORST SALZWEDEL	
IBM PC (8088)	4.77	BASIC INTERPRETER	2.0		890.000		J. SPEISER	DOUBLE PREC
CROMEMCO Z-80	4.0	32K STRUCTURED BASIC			900.000		A. ROXBURGH	DOODLE I REC
CROMEMCO Z-80	4.0	16K BASIC	5.70		900.000		A. ROXBURGH	
Z-80	2.0	NEUADA FORTRAN	3.0		975.000		SHERMAN GROMME	
SINCLAIR ZX-81	3.5	BASIC	5.0		990.000		PEARSON/CAIRO	"FAST" MODE
8085	6.0	CB-80	1.2		997.000		COMPUSERVE	11101 11000
IBM PC (8088)	4.77	DESMET C	2.2		1149.000		JEFF FURGAL	DOUBLE PREC.
8085	6.0	FORTRAN-80	3.4		1251.000		COMPUSERUE	DOUBLE I NEC.
BURROUGHS B20 (8086)		MS MULTIPLAN	3.1		1440.000		THOMAS MORAN	
NIPPON UNIVAC (8088)		T/MAKER III			1620.000		THOMAS MORAN	
8085		CBASIC	2.06		1623.000		COMPUSERUE	
ZILOG Z-80	6.0		2.00		1860.000		THOMAS MORAN	
	4.0	T/MAKER III BASIC-80	5.3		1980.000		COMPUSERUE	
Z-80 Z-80	4.0	AZTEC C II			2190.000		COMPUSERVE	
	U.T		1.05					
TI 99-4A (990X)	2.4	BASIC MC POCTO			2295.000		PEARSON/CAIRO	
TRS 100 (80C85)	2.4	MS BRSIC			2400.000		ALAN STEIN	
EAGLE II		CBASIC			2640.000		DTACK GROUNDED	
TI-59 CALCULATOR					3240.000		COMPUSERUE	
HP 41C	2 [0.00010			3500.000		K. FERGUSON	
Z-80	2.5	S-BASIC			3900.000		COMPUSERUE DONOLD LIBERTO	
HP 41CU CALCULATOR		FADTDON_DA			3920.000		RONALD WAGNER	DUIDI C DDCC
TANDY I (Z-80) TANDY I (Z-80)		FORTRAN-80			4602.000		KARL CASPER	DOUBLE PREC
Z-80	4.0	PASCAL-80 JRT PASCAL	2.1		5220.000 5760.000		KARL CASPER COMPUSERUE	
SINCLAIR ZX-81	1.0	BASIC			5760.000		PEARSON/CAIRO	"SLOW" MODE
HP-65 CALCULATOR		undic			6000.000		COMPUSERVE	JEUM HOUL
HP 15C CALCULATOR								
III TOC CHECULATUR					10200.000	1 11-7	KARL CASPER	

End Table

It's a headache, of sorts.

Whether your mailing list, accounting package, DBMS, or other enduser application is currently under development or already on the market, you've probably discovered that external sorting can be a difficult, costly, and time-consuming headache. Especially in today's floppy-disk storage environment.

T hat's why we developed
BEAMSORT™, the revolutionary inplace OEM sorting module. The older algorithms eat up valuable space on your user's often over-crowded diskettes. By installing BEAMSORT™ you get today's technol



CPM, MS-DOS, dBase II, and SuperSort are trademarks of Digital Research, Microsoft, Ashton-Tate, and Micropro, respectively. BEAMSORT™ runs under CP/M-80, CP/M-86*, MS-DOS,* and PC-DOS, supports multiple-volume files, ASCII, Microsoft .RAN, and dBASE II* .DBF file formats, and interfaces to all major languages. Custom interfacing, operating environments, and file formats are available.

What about performance?
BEAMSORT™ runs SuperSort*'s
own benchmark faster than SuperSort*
does! It's amazing, but don't take our
word for it. Write us on your company
letterhead for our OEM evaluation kit.
You must see this for yourself!

For fast relief.



Phlexible Data Systems, Inc.

3017 Bateman Street, Berkeley, CA 94705 (415) 540-7181

Circle no. 43 on reader service card.

TOTAL CONTROL:

FORTH: FOR Z-80[®], 8086, 68000, and IBM[®] PC Complies with the New 83-Standard

GRAPHICS • GAMES • COMMUNICATIONS • ROBOTICS DATA ACQUISITION • PROCESS CONTROL

- **FORTH** programs are instantly portable across the four most popular microprocessors.
- FORTH is interactive and conversational, but 20 times faster than BASIC.
- **FORTH** programs are highly structured, modular, easy to maintain.
- **FORTH** affords direct control over all interrupts, memory locations, and i/o ports.
- FORTH allows full access to DOS files and functions.
- **FORTH** application programs can be compiled into turnkey COM files and distributed with no license fee.
- **FORTH** Cross Compilers are available for ROM'ed or disk based applications on most microprocessors.

Trademarks: IBM, International Business Machines Corp.; CP/M, Digital Research Inc.; PC/Forth + and PC/GEN, Laboratory Microsystems, Inc.

FORTH Application Development Systems include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities and 200 page manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M II, \$100.00; 8080 FORTH for CP/M 2.2 or MP/M II, \$100.00; 8086 FORTH for CP/M-86 or MS-DOS, \$100.00; PC/FORTH for PC-DOS, CP/M-86, or CCPM, \$100.00; 68000 FORTH for CP/M-68K, \$250.00.

FORTH + Systems are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly.

PC FORTH + \$250.00 8086 FORTH + for CP/M-86 or MS-DOS \$250.00 68000 FORTH + for CP/M-68K \$400.00

Extension Packages available include: software floating point, cross compilers, INTEL 8087 support, AMD 9511 support, advanced color graphics, custom character sets, symbolic debugger, telecommunications, cross reference utility, B-tree file manager. *Write for brochure*.





Laboratory Microsystems Incorporated Post Office Box 10430, Marina del Rey, CA 90295 Phone credit card orders to (213) 306-7412



WHY FORTH?

- Genuinely Interactive (BASIC is much less interactive)
- Encourages Modular Programs (inefficiency and cluttered syntax hamper effective modularization in compiled languages)
- Fast Execution (not even C is faster)
- Amazingly Compact Code
- Fast Program Development
- Easy Peripherals Interfacing

HS FORTH

- Fully Optimized & Tested for: IBM-PC IBM-XT IBM-JR COMPAQ EAGLE-PC-2 TANDY 2000 LEADING EDGE and all MSDOS compatibles
- Graphics line, rectangle, block
- Music foreground and background
- Scaled decimal floating point
- Includes Forth-79 and Forth-83
- Full Support for DOS Files, Standard Screens and Random access DOS Screen Files
- Full Use of 8088 Instructions (not limited 8080 conversion subset of transported versions)
- Separate Segments for Code, Stack, Vocabularies, and Definition Lists - multiple sets possible
- Segment Management Support
- Full Megabyte programs or data
- Coprocessor Support
- Multi-task, Multi-user Compatible
- Automatic Optimizer (no assembler knowledge needed)
- Full Assembler (interactive, easy to use & learn)
- Compare BYTE Sieve
 Benchmark jan83
 HS/FORTH 47 sec BASIC 2000 sec
 w/AUTOOPT 9 sec Assembler 5 sec
 other Forths (mostly 64k) 70-140 sec
 PS You don't have to understand
 this ad to love programming in
 HS/FORTH!

HS/FORTH with AUTO-OPT & MICRO-ASM \$220.

Visa Mastercard Add \$10. shipping and handling

HARVARD SOFTWORKS

PO BOX 339 HARVARD, MA 01451 (617) 456-3021

C/UNIX PROGRAMMER'S NOTEBOOK

By Anthony Skjellum

In this column, we'll discuss some of the feedback received in response to previous columns, as well as some miscellaneous remarks.

uucp (Unix to Unix Copy)

Mike Meyers of Norman, OK, suggested that I publish my uucp address. Via this address, users on other Unix systems can send me mail electronically by using the standard Unix mail command. A uucp address is specified relative to a well-known site. Thus, the address "ucbvax | cithep | tony" should allow most users to reach me. If your site cannot communicate with ucbvax (UC Berkeley) directly, more site names must be prepended onto this address and onto those listed below as well. On the other hand, if your site can communicate with cithep (Caltech High Energy Physics) directly, you may omit the ucbvax prefix.

If a continuing electronic dialog develops, I will occasionally publish the uucp addresses of those involved so that others may join in. At the moment, the following users are participating in a discussion about Unix:

ucbvax|mtxinu|ea|jab
(Jeff A. Bowles)
ucbvax|cithep|yekta
(Yekta Gursel)
ucbvax|mtxinu|ea|emjej
(James Jones)
ucbvax|mtxinu|ea|mwm
(Mike Meyers)

I will include interesting remarks from this discussion in future columns for the benefit of those readers who cannot access the electronic mail.

C Users Group

Phillip K. Landis of Satellite Beach, FL, inquired about the address of the C Users Group of Yates Center, KS, which I mentioned in the April col-

umn. Their address and telephone are:

C Users Group 103 E. Rutledge Yates Center, KS 66783 (316) 625-3554

This particular group has 34 volumes of C programs and offers them in a variety of disk formats (only for microcomputers). I would like to print the addresses of any other public domain C repositories. If you know of one, please forward me their address.

Public Domain C

Robert E. Fiorini of Albany, NY, writes: "I'm a new C user trying to find an inexpensive (possibly public domain) C compiler (with source code ... if possible). My own efforts have been fruitless. I'm hoping that you or one of your readers may know where I can find such a compiler. If you can help me in any way ... please HELP!! ... It's funny, the only thing holding me back from the world of C is the C compiler itself."

Rod Cain's Small C is available through the C Users Group listed above. An enhanced version (not public domain) for 8080/Z80 CP/M systems is available (with source code) from The Code Works of Goleta, CA. (You can find their address in any number of DDJ back issues.) This particular compiler (Q/C) is a serious subset implementation and includes full compiler source for \$95.00. An IBM version of this product should be available later this year and would be well worth the investment if priced comparably to the CP/M version. For \$25 you can get version 2.1 of the Small-C compiler from J. E. Hendrix at Box 8378, University, MS 38677-8378 (see page 60 of the May 1984 DDJ for details). An MSDOS (IBM PC) version of Small C v.2.0 is available for \$35 from The Coriolis Company, Box 76, Clinton Corners, NY 12514.

Comments about C Input-Output

Mike Meyers writes: "What you haven't seemed to realize is that almost every flaw in Unix also appears in C. C is terse, doesn't protect the user, and is poorly documented. The only documentation for C is K&R [Kernighan and Ritchiel, which may be well written but is vague and inconsistent on all the points you turn to when you start implementing the language on new machines. To make matters worse, nobody (and I do mean nobody) sells a compiler that conforms to K&R, not even AT&T. I don't think anybody ever has, in any case. AT&T distributes a version of pcc [portable C compiler] that met K&R internally, but I think that by the time it was released externally, C had grown past K&R."

Gerald I. Evenden of N. Falmouth, MA, responded about C input-output as follows, with quite a different point of view:

"I was very disturbed with a basic concept about the C programming language that you kept implying in your column in the April issue. First of all I suggest that you carefully read the beginning paragraph of chapter 7 of Kernighan and Ritchie's book: The C Programming Language. It begins with 'Input and Output are not part of the C Language What remains is a description of I/O procedures contained in a standard Unix library, which will take care of most filter types of functional operations. I have generally found them to be quite adequate for most programming efforts involving stream data and simple questionanswer types of console I/O."

I would like to state that I am fully aware of the distinction between the C language definition and the standard input-output library. When teaching students how to program in C, this is

one of the first points I emphasize: C is a language that shows no special favoritism to a specific set of input-output routines. (A single standard set does exist, and this is the Unix standard.) I consider input-output independence an essential feature of C, but I feel that a discussion of a real C compiler environment cannot always be separated from a discussion of the support library that comes with it. I also maintain that the Unix input-output library is more than adequate for dealing with stream operations. Mr. Evenden summarizes the distinction between C and C input-output as follows:

"The beauty of C is that it doesn't have a plethora of specialized built-in functions but rather provides the programmer with a rich facility to build tools required for his own, occasionally specialized, needs. Obviously, we shouldn't have to redesign all the wheels needed, so most suppliers of C compilers include a library of functions patterned after the Unix libraries. But remember, there is absolutely no requirement to use them if they don't fit your needs, and they should only be viewed as a preliminary tool kit."

One point that merits further exploration is that of portability. While it is well and good to preach the separation of C and C input-output, only software that uses standard Unix input-output calls (and routines built on them) has a prayer of being moved readily between different machines or even between different compilers on the same machines.

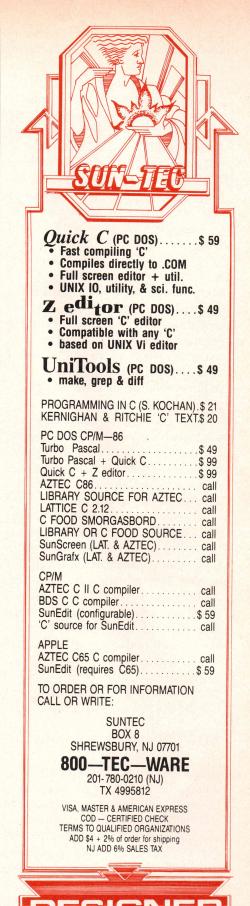
Mr. Evenden continues: "I suspect that your problem with 'getc' et al. is related to screen editing and control, which is a category of program that doesn't fall into the 'filter' class of functions emphasized by Unix (and its libraries), and I certainly agree that these functions don't work in this case The astute programmer writes 'rawin()' and 'rawout()' to satisfy those needs. There's nothing to prevent it and everything to encourage it The worst possible outcome of the problems posed in your article is to even remotely suggest rewriting the current stream I/O functions. Their current form is a de facto standard, and a consistency of implementation is expected by most C programmers."

I really don't expect anyone to throw away the existing stream functions. Not only would this be unreasonable, it would also be undesirable. However, I do feel that clean raw input-output should be a supported capability; it needn't be reinvented each time a Unix programmer discovers that stream input-output is inconvenient for interactive purposes. In discussing a similar worry expressed by Mr. Meyers, I summarized my argument by stating that interactive programs comprise a large fraction of those run by Unix users, and that when programmers write interactive programs, they don't usually want the users treated like input

Despite Mr. Evenden's outspoken letter, we actually agree in many respects. However, some things he brought up demand careful examination. He states:

"The principal point of this complaint is that you should be a little more careful of what you are talking about. Writing about problems with C I/O is impossible since C I/O doesn't exist. However, your less-experienced readers will take your complaint to heart and decide that C is a useless language because Mr. Skjellum, et al., don't like the optional I/O library supplied with their compiler. If you were more positive in your approach you would be telling readers how to write their own procedures to do specialized console I/O on Unix, CP/M, etc. I've done it on both CP/M and Unix and found it to be a piece of cake in both cases and never gave the 'getc' group a second thought when it was obvious that they were not meant for the job at hand."

The comments I have made are based on several years of experience with C under Unix, CP/M, VMS, etc. One must come to grips with reality. C input-output, while optional, is normally what users must utilize to deal with a problem at hand; consequently, inexperienced users must lean more heavily on the standard library than experienced users. It is meaningful to discuss C input-output. It does exist, and Mr. Evenden discusses several points about it before stating that the topic is beyond the realm of discussion. Inexperienced users learn by reading dialog between others who have seen



Circle no. 35 on reader service card.

problems in their own work. Censoring this information to "protect" such users from disenchantment with C is an unacceptable alternative.

We have not reached the computer millenium. C and Unix as existing tools have flaws and drawbacks. Only through discussion can we seek solutions and create better future systems. The idea of restricting discussions because of semantic points seems to be contrary to that goal. Some other writers have taken the approach that C and Unix are wonderful tools and heap praise on them in review after review. A certain group of individuals feel highly insulted if this approach is not followed. In an evolving field, it makes sense to criticize as part of the learning process. That is why I include Mr. Evenden's final remarks, because I think that he has drawn a counterproductive conclusion from an understandable point of view:

"C is not a perfect language but it certainly beats what's in second place. Consequently, my enthusiasm about C makes me very chauvinistic about misplaced and invalid criticisms. I have a couple of minor complaints about some aspects of C but I bite my tongue when I think about the dark ages. After several happy years with ALGOL in the 60s I was sentenced to over 10 long years of Fortran purgatory before being born again with C. I guard this language jealously and you had better be careful of what you write or I'll curse you to a task of debugging 10,000 lines of BASIC code."

I hope that Mr. Evenden will reconsider and send in his comments about C so we can add them to the discussion.

BDS C Runtime Solution

Alex Cameron of Malvern (Victoria), Australia, had the following comments: "I couldn't help responding to your notes on the nonstandard nature of some of BDS C's runtime routines (*Dr. Dobb's* No. 90). There is probably little doubt that most of us gladly suffer its irregularities because of its speed, low price, and because it is arguably one of the finest C compilers around—all this notwithstanding, I still find the nonstandard buffered file functions such as fopen the most frustrating, simply because of the need to

continually declare buffers."

The listing on page 120 (listing stdlib3.c) is Mr. Cameron's proposed solution to this problem under BDS C.

Unix Comments

Two users had responses to previous comments about Unix. Tim Prince of Marblehead, MA, writes:

"As a new Unix user I find some of the subjects raised in your column this month [April] interesting. Certainly, in comparison with VAX/VMS, there is a lack of discipline in adhering to uniform standards for a user interface. This is the almost inevitable result of the way Unix evolved as the product of the work of various organizations. Also the volume of official documentation is much less, but if you count the many good books available from third parties, the comparison will soon go the other way. . . . Unix has collected a set of useful information into a pair of volumes, which the individual can afford to own."

He adds: "I have spent nearly 20 years learning things about GCOS, which has all the faults of which Unix is accused without many of the advantages. This operating system will soon disappear from the face of the earth, making our knowledge of its quirks so much nonbiodegradable mental trash. What is learned about Unix is more likely to remain useful after the current hardware is gone."

Mike Meyers writes: "... I tend to agree with most of the things you've said about Unix. The documentation is, at best, terse. What's worse, it's either missing or wrong in many places. To top it off, most of it assumes that you have access to the source and are a good computer scientist. To make matters still worse, there really isn't an online documentation system. What you have is an interactive manual printer. It is very definitely not friendly to the naive user. The best description I have run across is that Unix is expertfriendly. If you know it, it's going to be great. If you don't know it, well, that's just too bad."

These two viewpoints are not mutually exclusive. Unix is expert-friendly and beginner-hostile; many of the Unix functions such as man and mail are primitive, but this bothers some us-

ers less than others.

One can argue that, like C inputoutput, specific Unix tools are optional and can be replaced by programs preferable to the user. By this reasoning, it is not meaningful to complain about specific Unix features/programs. However, I feel that most users want to take Unix facilities off the shelf and spend their time on an application, not on reworking existing software tools. Therefore, comments about deficiencies in specific Unix tools are also germane.

Conclusion

This column exists for the purpose of discussing C and Unix as they are, with the problems they have. As stated emphatically above, C and C inputoutput are independent concepts. However, users generally deal with the input-output library, and deviations from that library have given users cause for complaint. Therefore, such a discussion has a place here. I invite readers to submit any code that illustrates how to deal with raw input-output in C under CP/M, Unix, or other operating systems.

In this column, we have considered some user comments about C/Unix based on several letters received recently. I would be interested to hear what others think about this ongoing discussion, as well as follow-up comments from those readers represented here.

Reader Ballot

Vote for your favorite feature/article.

Circle Reader Service No. 197.

C/UNIX Listing (Text begins on page 116)

```
/*
** stdlib3.c -- standard I/O library
**
** Copyright 1984 A. Cameron
*/
#include "bdscio.h"
/*
**
        Standard fopen
**
**
              return fd on success, NULL on error
**
**
*/
sfopen(filename, mode)
char *filename;
char *mode;
(
         int fd;
         if (*mode == 'w')
                                 /* write mode */
                 if (!(fd = alloc(BUFSIZ)))
                         return(NULL);
                 else
                          if (fcreat(filename, fd) == -1)
                                  free(fd);
                                  return(NULL);
                          >
                         return(fd);
                 3
        3
        if (*mode == 'r')
                                  /* read mode */
        (
                 if (!(fd = alloc(BUFSIZ)))
                         return(NULL);
                 else
                 {
                          if (fopen(filename, fd) == -1)
                                  free(fd);
                                  return(NULL);
                          3
                         return(fd);
                 3
        3
```

C/UNIX Listing (Listing continued, text begins on page 116)

```
if (*mode == 'a')
                                  /* append mode */
                 if (!(fd = alloc(BUFSIZ)))
                         return(NULL);
                 else
                 {
                         if (fappend(filename, fd) == -1)
                                  free(fd);
                                  return(NULL);
                          >
                         return(fd);
        3
        return(NULL); /* failure */
3
/*
**
        Standard follose
**
**
*/
sfclose(fd)
int fd;
(
        fputc(CPMEOF, fd);
        if (!(fclose(fd)))
                 free(fd);
                 return(NULL);
         3
        return(ERROR);
3
```

End Listing

Re-ink any fabric ribbon for less than 5¢. Extremely simple operation. We have a MAC INKER for any printer. Lubricant ink safe for dot matrix printheads. Multicolored inks, uninked cartridges available. Ask for brochure. Thousands of satisfied customers.

\$5495 +

Mac Switch lets you share your computer with any two peripherals (serial or parallel). Ideal for word processors—never type an address twice. Ask us for brochure with tips on how to share two peripherals with MAC SWITCH. Total satisfaction or full refund.

\$9900

Climputer





Order Toll Free 1-800-547-3303

MacInker Suite #10 Portland, Oregon 97225 (503) 297-2321 Reference to the suite of the suite

Circle no. 12 on reader service card.

GET "C" APPLICATIONS OFF TO A FLYING START WITH

C-TREE TM

C-SORT TM

RECORD MANAGEMENT SUBSYSTEM

- · Advanced B + Tree Structure
- · Fast And Efficient
- · Unlimited # Of Keys
- Keys Mav Be Duplicate, LIFO/ FIFO, Modifiable
- · Record Locking Calls
- · Sequential Access
- Utilities To Add/Delete Keys And Fields, Rebuild Files
- Error Processing Interface
- Store Data Dictionary In File

ordering information SINGLE UNIT LICENSE

\$99 per program plus shipping. Format 51/4 Disk MS-DOS Compatible Linkable 8086-file format modules for Lattice-C Compliers, others soon. Complete documentation.

AccuData Software

Dept. D-8
P.O. Box 6502

Austin, Texas 78762

SORT/SELECT/MERGE SUBSYSTEM

- Advanced Quick/Tournament Sort
- Sort B-Tree or Sequential Files
- Automatically Uses All
 Available Memory
- Sort On Any Number/Type Of Field
- Select Records According To User-Specified Criteria
- · Creates Tag (Index) Sorting File
- Automatic Interface To B-Tree

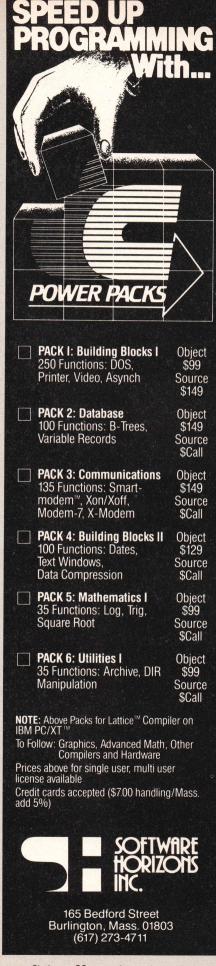
SOURCE CODE OPTION

\$249 per program plus shipping. "C" Source Code is also available: requires license. A credit is allowed for object license purchased previously.

MULTIPLE COPY OPTION

Multiple copies of object code may be made with this license at a very low unit cost.

> Telephone Orders Accepted Visa/Mastercard (512) 476-8356



SOFTWARE REVIEWS

PC-Write

Company:

Quicksoft, 219 First N. #224, Seattle, WA 98109 Computer: IBM PC Price: \$10; Full registration \$75 Circle Reader Service No.127 Reviewed by Ronald G. Parsons

The number of good programs available for the IBM PC under marketing arrangements similar to Freeware is increasing. This is a good sign for both users and the industry as a whole. Users are able to obtain good, if not the best, software at a reasonable price, and the small software developers are able to gain some market share without requiring a million dollar advertising campaign to get started. PC-Write is a good example of this trend.

Bob Wallace, PC-Write's designer, is making it available under a variation of Freeware he calls commission Shareware. If you register your copy of PC-Write with Quicksoft and someone obtains a copy from you and registers it, Quicksoft will give you a commission of \$25, one-third of the registration fee. Register your copy, give away three copies, and, when they are registered, you will have all the advantages of registration at no cost. Registration has several benefits: telephone support, a bound copy of the documentation, and the complete source code (Pascal and assembler).

PC-Write provides a full set of word processing functions without a lot of the frills that may or may not be useful to many users. All the functions necessary for editing letters and short documents are provided. However, a way to merge several files together is only weakly provided, and there is no spelling checker. I consider these two features to be extremely valuable—almost mandatory. The program does create standard PC-DOS files so that many standard spelling checkers will

work, including IBM's Word Proof, which I consider the best value available today in PC software.

Printing functions are provided with a separate program so that true "what you see is what you get" is not available. Support is provided only for dumb printers, although users can imbed printer control escape sequences and control codes if they desire; that does not come under the heading of ease-of-use. Headers, footers, margin and spacing control, page skipping, and numbering, as well as input file changing, are about all that is provided. For much work this is all that is needed.

All the function keys, Shifted, Alted, and Ctrl-ed, are used, as are many control keys. The functions assigned to the control keys, which are user configurable, are initially set to be similar to WordStar. Configuration is a strong point of PC-Write. The use of color (where available) for different word processing functions is well done. Pressing F1 produces a menu of all the function keys and other major operations. However, so many key functions are available that the help screen is bewilderingly complex. The designer seems to have leaned toward providing more function at the expense of simplicity and ease-of-use. I would hesitate to give this program to users unaccustomed to the complex keyboard of the IBM PC and expect them to be able to use PC-Write quickly. As an example, PC-Write has at least six or eight different cursor shapes—each denoting the present status of some function, such as insert (or pushright) mode, shift or control key status, or the current status of marking text. The variations are too many and too subtle for my taste.

Cursor movement is available in many ways—character, line, word, paragraph, or screen (up or down). However, the convention of which key to use is reversed from many other PC

products. The meaning of the PageUp and PageDown keys can be reversed, but others cannot. All this leads to more confusion for inexperienced users.

The only bug I ran into was an inability to handle files that have line-feeds without always being accompanied by a carriage return. This interferes with the paragraph formatting and line delete functions and causes the program to loop, requiring a power-off reset to continue.

I found the product to be useful but too complex. In my rating scheme, ease-of-use is the most important attribute. The manual provided is well written, but it is hard to find information quickly. Topic headings in larger typefaces would be desirable.

PC Small-C 2.0

Company:

The Coriolis Company Box 76, Clinton Corners, NY 12514

Computer: IBM PC Price: \$35

Circle Reader Service No. 125
Reviewed by Ronald G.Parsons

Over the years, Dr. Dobb's Journal has fostered and published the source code for a number of implementations of small versions of the C programming language. Ron Cain developed the original Small-C compiler for the 8080 microprocessor, published in Dr. Dobb's Journal in May 1980 and September 1980. Versions of this compiler were made available for CP/M (The Code Works) and IBM PC-DOS 1.1 (Caprock Systems). J. E. Hendrix developed a later version, Small-C 2.0, published in Dr. Dobb's Journal in December 1982 and January 1983. The product described in this review, PC Small-C 2.0, is a revision of Small-C 2.0 for IBM PC-DOS and Microsoft MS-DOS (versions 2.0 and later).

PC Small-C 2.0 is not a complete implementation of the C language, nor does it claim to be. It does provide an inexpensive way to learn about and try the C language. It also provides, through the source code and libraries. an excellent way to learn about the structure and coding of high-level language compilers. The minimum requirements for using PC Small-C are an IBM PC (or equivalent) with 96K memory, one single-sided diskette drive, PC-DOS or MS-DOS 2.0 or later. and the 8088 PC Macro Assembler.

PC Small-C provides certain enhancements over Small-C 2.0, including redirection of input and output, additional library functions, hexadecimal and octal numeric constants, code generation optimized for the 8088 microprocessor, and automatic declaration of external function references. The authors claim that PC Small-C 2.0 programs are typically 30% smaller and twice as fast as equivalent Caprock Small-C:PC programs.

Features

PC Small-C 2.0 supports character and integer data types as global, local, external, and pointer variables, as well as one-dimensional arrays. Not supported are float, double, unsigned, long, short, or register variables, structures, or higher dimensional arrays. All the usual unary and binary operators are supported, as well as the selection operator. Statements supported include if, if ... else, while, do ... while, for, switch/case/default, break, continue, goto, and return. Compiler directives supported include #define, #include, and #ifdef ... #else ... #endif. In-line assembler code is supported by the #asm ... #endasm directive. A fairly complete set of I/O library routines is included, but open. close, read, and write system calls are not provided. This may cause some inconvenience in transporting C programs to this compiler. Many useful miscellaneous library routines include abs, isalpha, islower, isupper, peek, poke, tolower, and toupper. Several IBM PC-specific routines are also included, such as delay, sound, spkroff, and spkron. Command line argument support is available through argc, argv, and getarg.

	Small-C 2.0	Computer Innovations'	Lattice C
Compile/Link	66 sec	Optimizing V2.10 49 sec	version 2.00 46 sec
Execute	36 sec	14 sec	12 sec
.EXE size	14,369 bytes	16,015 bytes	19,618 bytes

Figure

PRICE BREANTHROUGH The wait-loss experts have done it again!

512Kbyte SemiDisk with SemiSpool

Time was, you thought you couldn't afford a SemiDisk. Now, you can't afford to be without one.

	256K	512K	1Mbyte
SemiDisk I, S-100	\$895	\$1095	\$1795
IBM PC		\$1095	\$1795
TRS-80 Mdl. II, CP/M		\$1095	\$1795
SemiDisk II, S-100		\$1395	\$2095
Battery Backup Unit	¢150		

Version 5 Software Update \$30

Time was, you had to wait for your disk drives. The SemiDisk changed all that, giving you large, extremely fast disk emulators specifically designed for your computer. Much faster than floppies or hard disks, SemiDisk squeezes the last drop of performance out of your computer.

Time was, you had to wait while your data was printing. That's changed, too. Now, the SemiSpool print buffer in our Version 5 software, for CP/M 2.2. frees your computer for other tasks while data is printing. With a capacity up to the size of the SemiDisk itself. you could implement an 8 Mbyte spooler!

Time was, disk emulators were afraid of the dark. When your computer was turned off, or a power outage occurred, your valuable data was lost. But SemiDisk changed all that. Now, the Battery Backup Unit takes the worry out of blackouts.

But one thing hasn't changed. That's our commitment to supply the fastest, highest density, easiest to use, most compatible, and most cost-effective disk emulators in the world.

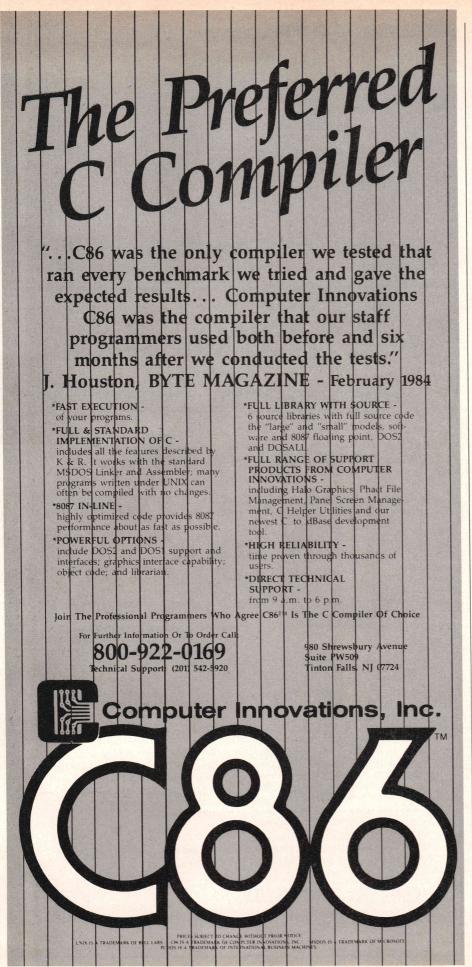
SemiDisk. It's the disk the others are trying to copy.

SEMIDISK SYSTEMS, INC.

P.O. Box GG Beaverton, OR 97075 (503) 642-3100

Call 503-646-5510 for CBBS*/NW, a SemiDisk-equipped computer bulletin board 300/1200 baud SemiDisk, SemiSpool Trademarks of SemiDisk Systems. CP/M Trademark Digital Research





Circle no. 13 on reader service card

Memory Utilization

PC Small-C 2.0 trades off memory availability (stack, data, and code are limited to a total of 64K) against speed of execution. For small C implementations, this is probably the proper tradeoff. Large programs requiring other data models should use a full implementation of C. In-line code sequences are favored for speed, and the use of segment override prefixes and the BP register is avoided. This is made possible by setting the data segment register, DS, and the stack segment register, SS, to the same value during initialization.

Compilation

The compilation process consists of a compiler phase, which produces assembler source code; an assembly phase, which produces an object module; and a link phase, which links the program object module with library object modules or separately compiled modules. Because of this assembly step and the slowness of the IBM assembler, the compilation process is somewhat lengthy. For instructional purposes, however, the easy availability of the generated machine code is invaluable. A compiler switch can cause the Small-C source code lines to be included as comments in the assembler source code.

Benchmark

No review of a C compiler seems to be complete without a benchmark using 10 iterations of the Sieve of Eratosthenes. The times in the figure (page 123) were obtained using an IBM PC XT running PC-DOS 2.1.

Conclusion

PC Small-C 2.0 is an excellent value to someone wanting to investigate or learn C or to learn how compilers are constructed. The small price includes a reference manual, the complete source and executable code, and even a remittance to Ron Cain and J. E. Hendrix.

helps compare, evaluate, find products. Straight answers for serious programmers

SERVICES

- Programmer's Referral List Dealer's Inquire Newsletter
- Help find a Publisher
- Evaluation Literature free
 Over 300 products
 BULLETIN BOARD 7 PM to 7 AM 617-461-0174

Our Free Report: PRODUCTIVITY - MSDOS

Assume use of compiler and typical editor. What commercial or public domain products, what techniques improve productivity? "Productivity with MSDOS" is a growing document with some answers. Call to request it. Help improve it. Earn \$50 credit toward any purchase when we add any description, code, or idea received from you

"C" LANGUAGE PRICE PRICE APPLE: AZTEC C - Full, ASM \$199 call 8080: BDS C - Fast, popular 8080: AZTEC C - Full Z80: ECOSOFT - Fast, Full 199 call 250 225 8086: C86 - optimizer, Meg 8086: Lattice - New 1.1 & 2.0 call

COOC. Lattico Hon I. I C	2.0	000	
Microsoft (Lattice) M	SDOS	500	C
Digital Research - Megabyte	8086	350	2
Desmet by CWare - Fast	8086	109	
DAGIO SHUDON	AFNIT		

ENVINO	MINICIAL		
Active Trace - debug	8080/86		72
MBASIC-80 - MicroSoft	8080	375	255
BASCOM-86 - MicroSoft	8086	395	279
CB-86 - DRI	CPM86	600	419
Prof. BASIC	PCDOS	345	325
BASIC Dev't System	PCDOS	79	72

FEATURES

- C INTERPRETERS for MSDOS Ask about one for beginners for \$85 or full development for \$500.
- C HELPER includes source in C for MSDOS or CPM80 for a DIFF, GREP, Flow charter, C Beautifier and others. Manage your source code easier. \$125.
- PROLOG86 Interpreter for MSDOS includes tutorials, reference and good examples. Learn in first few hours. For Prototyping, Natural Language or Al. \$125

EDITORS P			
C Screen with source	8080/86	NA	60
EDIX - clean	PCDOS	195	139
FINAL WORD - for manu	als 8080/86	300	215
MINCE - like EMACS	CPM, PCDOS	175	149
PMATE - powerful	CPM	195	175
	8086	225	195
VEDIT - full, liked	CPM, PCDOS	150	119
	8086	200	159

FORTRAN			
MS FORTRAN-86 - Meg	MSDOS	\$350	\$255
SS FORTRAN - 86	CPM-86		345
FORTRAN-80 - 66 decent	CPM-80	500	350
INTEL FORTRAN - 86	IBM PC	NA	1400
DR FORTRAN COMING			100
RM FORTRAN COMING			

LANGUAGE LIBF	ARIE	S	
C to dBASE interface	8080/85	\$125	\$115
C Tools 1 - String, Screen	PCDOS	NA	115
C Tools 2 - OS Interface	PCDOS	NA	92
FLOAT 87 - Lattice, PL1	PCDOS	NA	115
GRAPHICS: GSX - 80	CPM80	NA	75
HALO - fast, full	PCDOS	200	175
Greanleaf for C - full, 200+	PCDOS	NA	165
ISAM: Access Manager - 86	8086	400	300
BTRIEVE - many languages	PCDOS	245	215
PHACT - with C	PCDOS	NA	250
FABS	CPM80	150	135
PASCAL TOOLS - Blaise	PCDOS	NA	115
SCREEN: Display Mgr. 86	8086	500	375
PANEL-86 - many languages	PCDOS	295	245
WINDOWS for C	PCDOS	NA	139
Virtual Screen - Amber	PCDOS	295	call

Call for a catalog, literature, and answers

800-421-8006

THE PROGRAMMER'S SHOP™

128 DRockland Street, Hanover, MA 02339 617-826-7531, Mass: 800-442-8070

MasterCard

RECENT DISCOVERIES

PROFILER - Examine MSDOS program execution speeds. Determine where to improve programs in any Microsoft language, Lattice, or C86. Make histograms that show time spent in portions of your program, and doing MSDOS I/O, etc. \$175.

		LIST	UUK
PASCAL	ENVIRONMENT	PRICE	PRICE
PASCAL MT + 86	CPM86/PC	\$400	\$279
without SPP	CPM80	350	239
MS PASCAL 86	MSDOS	350	249
SBB PASCAL-great, fas	st PCDOS	350	315
PASCAL 64 - nearly full	COM 64	99	89
SBB Jr - best to learn	PCDOS	NA	95

OTHER PRODUCTS

CHARLES AND ADDRESS OF THE PARTY OF THE PART			
Assembler & Tools - DRI	8086	200	159
COBOL - Level II	8086	1600	1275
CODESMITH-86 - debug	PCDOS	149	139
GCLISP	PCDOS	495	475
IQ LISP - full 1000K RAM	PCDOS	175	call
Janus ADA - solid value	PCDOS	500	449
MBP Cobol-86 - fast	8086	750	695
Microshell improve CPM	8080	150	125
Microsoft MASM-86	MSDOS	100	85
PL/1-86	8086	750	530
PLINK-86 - overlays	8086	350	315
POWER - recover files	8080/86	169	139
READ CPM86 from PCDOS		NA	55
READ PCDOS on an IBM PC	CPM86	NA	55
Trace 86	PCDOS	125	115

Note: All prices subject to change without notice. Mention this ad. Some prices are specials

Ask about COD and POs All formats available

Circle no. 47 on reader service card.

Prolog-86

Learn Fast. **Experiment, Prototype**

1 or 2 pages of PROLOG would require 10 or 15 pages in "C"- and PROLOG is much easier.

In one evening develop a feel for PROLOG. In a few days understand and enhance artificial intelligence programs included like:

> * an Expert System * a Natural Language Processor

PROLOG-86 includes two tutorials, a reference, 6 sample programs and a PROLOG Interpreter.

Intro price: \$125 for PCDOS, MSDOS or CPM-86. Call:

Full Refund if not satisfied in first 3 weeks.

SOLUTION SYSTEMS**

45-D Accord Park Drive, Norwell, MA 02061

617-871-5435

C Helper

UNIX[™]-like Utilities for C Programming with source

Save time when working with your C programs. Full source lets you make them work your way, helps you learn.

Utilities included: compare files (DIFF), cross reference your variables (CCREF), examine the flow of functions as they call each other (FCHART), format and indent programs (pretty printer), search for patterns (GREP). Others check program syntax, print programs your way and more.

UNIX is a trademark of Bell Labs.

\$135 for PCDOS, MSDOS, CPM-86 or CPM80.

Call with questions or for "Programming with C Helper".

SOLUTION SYSTEMS

45-D Accord Park Drive, Norwell, MA 02061

617-871-5435

by Michael Wiesenberg

Turn jr into Papa

The **Drive Two Enhancement Package** from Rapport Corporation is an expansion kit for the IBM PCjr that enables it to run "virtually all" IBM PC software. For \$675, you get a 360K floppy disk drive, parallel printer port, clock calendar with battery backup, and 128K memory (expandable to 512K). Optional accessories include an audio amplifier with speaker, to provide audio reproduction on any monitor, and a keyboard adapter cable. **Reader Service No. 101.**

Pascal on PC, jr, etc.

SBB Pascal, by Software Building Blocks, runs on the IBM PC and compatibles; with PC-DOS, MS-DOS, and CP/M-86; and also, they claim, on the PCjr. Execution time and code size compare favorably to other Pascal packages, according to benchmarks in their brochure. One of the best features is that you don't have to pay royalties on products using SBB code. The compiler package is \$95. The development package is \$350. The latter adds separate compilation, overlays, BCD arithmetic, 8087 support, sources for the runtime library, updates, and a copy of Watt's Pocket Guide to Pascal (Addison-Wesley). You get a debugger, editor, and editor source with both versions. Reader Service No. 103.

And Modula-2 for PC XT

Volition Systems, with versions of **Modula-2** already out for Apple, Sage, and IBM PC, now offers the language specifically for the XT—with utilities, access to any memory location, variables based at fixed memory addresses, RAM disk support, access to I/O ports, screen display

mode, 8087 coprocessor support, exception handling, subprogram calls and overlays, random and sequential file access, directory operations, concurrency and interrupt handling, strings, and BCD arithmetic. For \$395 you get both Modula-2 and Pascal compilers, a module library, the Advanced System Editor (ASE) for program development and text processing, a p-shell (Unix-like programming environment), utility programs, complete documentation, an overview for new users, a tutorial for those familiar with Pascal, installation guide, ASE User's Manual, programming hints, and Wirth's Programming in Modula-2. Reader Service No. 105.

C64 Disassembled

C-64 source, from Schnedler Systems, is the complete assembly language source for the Commodore 64's BASIC and Kernel ROMs, cross referenced and extensively commented. Even included are calls to routines not found in Commodore documentation. \$29.95. Reader Service No. 113.

Cross-Assemble Almost Anything

2500 AD Software has cross assemblers that run on MS-DOS, PC-DOS, CP/M-80, CP/M-86, and Apple II with softcard. You can get all major 4- and 8-bit families, including F8/ 3870, COPS 400, NEC 7500, 87xx, 8051, 68xx, 65xx, 1802, NSC-800, Z-8, Z80, 8080, and 8085, and 16- and 32-bit microprocessors, including 8086/88, 80286, 16032, Z8000, and 6800. All packages include relocatable code, large file-handling capacity (buffer overflow to disk), macros, recursion, nesting, conditional assembly (to 248 levels of nesting), assembly time calculator, "include" files, listing control, hex file converter, cross reference table, and plain English error messages. \$99.50 for 4- and 8-bit products, and \$199.50 for 16- and 32-bit (plus \$6.50 p. & h. each). Reader Service No.109.

Full C for PC

c-systems has a full C compiler for the IBM PC and MS-DOS computers, supporting 8087 or assembly language subroutine floating point, and it addresses up to 1 Mb code or data. You get transcendental functions, ROMable code generation, full featured liraries (including DOS 2.0 functions with automatic selection of DOS 1.1 and 2.0 calls), and program chaining. You also get c-window, a sourcelevel debugger that displays source code while debugging, variable display and alteration, automatic command execution, and multiple breakpoints that can be set as function and line number. The package also includes a fast assembler. All of this will set you back but \$199. Reader Service No. 107.

Keep it Clean

Mini-Vac keeps your computer running longer by removing dirt from between the keys and other inaccessible areas. You can also clean stereo, video, and other electronic equipment, cameras, and other areas most easily reached by a portable vacuum cleaner. This is not a compressed air unit that merely disperses pollutants; it sucks them into a replaceable cloth vacuum bag. You can also use Mini-Vac as a blower. It has two wands, two bristle brushes, and can be powered by DC or AC, with an optional adapter. It costs \$29.95 and has a 90-day guarantee. Reader Service No. 115.

On-Line Desktop Accessories

Ever get a phone call in the middle of a computing session, need to jot down a note, and can't find paper and pencil? Or have to stop in the middle of editing to make a computation and can't find your calculator? Sidekick from Borland opens windows in your application program with electronic notepad and editor, calculator, appointment calendar, ASCII table. automatic dialing, metric conversion table, on-line help, and customizing capabilities to bring in special tools. You get into or out of Sidekick with a single keystroke. Currently available for IBM PC, XT, PCir, and "true" compatibles, the product costs \$49.95. Reader Service No. 111.

Moments to Remember

The Society for Computer Simulation holds its annual Summer Computer Simulation Conference July 23–27 at the Copley Plaza Hotel in Boston. Topics include continuous systems simulation languages, desktop simulation, supercomputers, artificial intelligence, and, in a special meeting of the Dutch Benelux Simulation Society, research topics on advanced information processing for simulation in the Benelux countries and demonstration of the Delft parallel processor. Conference registration is \$155 for participants and members of sponsor or affiliate societies and \$170 for nonmembers. Reader Service No. 117.

The American Association for Artificial Intelligence and the University of Texas at Austin Departments of Mathematics and Computer Sciences are jointly sponsoring a conference on artificial intelligence research August 6-10 at the Performing Arts Center at the University of Texas at Austin. Reader Service No. 119.

The Association for Computing Machinery (ACM) holds its 1984 annual conference October 8-10 at the San Francisco Hilton. The main topic of discussion will be the fifth generation challenge from Japanese supercomputers. Reader Service No. 121.

The Sixth Annual Forth Conven-

tion, held by the Forth Interest Group (FIG), takes place November 16-17 at the Hyatt Palo Alto, with tutorials, exhibits, vendor booths, lectures, and discussions. FIG is a worldwide, nonprofit organization that has over 4700 members and 53 chapters. Reader Service No. 123.

Contact Points

ACM-84, The Fifth Generation Challenge, Box 32575, San Jose, CA 95152; (415) 948-6306.

American Association for Artificial Intelligence, 445 Burgess Drive, Menlo Park, CA 94025; (415) 328-3123.

Borland International, 4113 Scotts Valley Drive, Scotts Valley, CA 95066; (408) 438-8400.

c-systems, Box 3253, Fullerton, CA 92634; (714) 637-5362.

Forth Interest Group, Box 1105, San Carlos, CA 94070; FIG Hot Line (415) 962-8653.

Mini-Vac, Inc., Box 3981, Glendale, CA 91201; (213) 244-6777.

Rapport Corporation, 80 South Redwood Road, Suite 213, North Salt Lake, UT 84054; (801) 292-9454.

Society for Computer Simulation, Box 2228, La Jolla, CA 92038-2228; (619) 459-3888.

Software Building Blocks, Inc., Box 119, Ithaca, NY 14851-0119; (607) 272-2807.

Schnedler Systems, 1501 N. Ivanhoe, Dept. NR, Arlington, VA 22205: (703) 237-4796.

2500 AD Software, Inc., 17200 East Ohio Drive, Aurora, CO 80017; (303) 752-4382.

Volition Systems, Box 1236, Del Mar, CA 92014; (619) 481-2286.

DDJ

Reader Ballot Vote for your favorite feature/article. Circle Reader Service No. 198.

DeSmet The fastest 8088 C Compiler available

FULL DEVELOPMENT PACKAGE

- C Compiler
- Assembler
- Linker and Librarian
- Full-Screen Editor
- Newsletter for bugs/updates

SYMBOLIC DEBUGGER

- Monitor and change variables by name using C expressions
- Multi-Screen support for debugging PC graphics and interactive systems
- Optionally display C source during execution
- Breakpoint by Function and Line #

COMPLETE IMPLEMENTATION

- Both 1.0 and 2.0 DOS support
- Everything in K&R (incl. STDIO)
- Intel assembler mnemonics
- · Both 8087 and Software Floating Point

OUTSTANDING PERFORMANCE

Sieve Benchmark

COMPILE 4 Sec. RAM -

22 Sec FDISK

6 Sec. RAM -LINK

34 Sec. FDISK

RUN 12 Sec.

SIZE 8192 bytes

DeSmet C Development Package \$159

To Order Specify:				
Machine				
OS ☐ MS-DOS ☐ CP/M-86				
Disk □ 8" □ 51/4 SS □ 51/4 DS				
WARE				
CORPORATION				

P.O. BOX 710097 San Jose, CA 95171-0097 (408) 736-6905

California residents add sales tax. Shipping: U.S. no charge, Canada add \$5, elsewhere add \$15. Checks must be on a US Bank and in US Dollars.

Circle no. 17 on reader service card.

ADVERTISER INDEX

Read			Read			Read	
Servi	ce	Page	Servi		Page	Servi	
No.	Advertiser	No.	No.	Advertiser	No.	No.	Advertiser No.
			22	<u>.</u>	16	48	PRO Microsystems113
1	AccuData Software		22	Faircom		49	Protools
2	Amber Systems		23	Farbware		50	Ouelo
3	Application Executive Corp		24	Foehn Consulting		51	Quest Research
4	Ashton-Tate		25	GGM Systems, Inc		52 -	
5	Avocet Systems, Inc		26	GTEK			Quick-N-Easi Products, Inc 36-37
6	BD Software		27	Harvard Softworks		53	Rational Systems, Inc
7	B.G. Micro		28	Integral Quality			Edward Ream
8	Borland International		29	Key Solutions		54	Revasco
9	Carousel Micro Tools	47	30	Laboratory Microsystems		55	Robert Jacob Agency
10	Compu-Draw	41	31	Lattice, Inc.		56	SemiDisk Systems
11	Compusophic Systems	33	32	Leo Electronics, Inc		57	SLR Systems
12	Computer Friends		33	Lifeboat Associates		58	Software Building Blocks 61
13	Computer Innovations	124	34	Logical Devices		59	Software Engineering Consultants . 61
14	Creative Solutions		35	Sun-Tec		60	Software Horizons, Inc
15	C Systems	24	36	McDermott Computer Services .	61	74	Solution Systems
16	C User's Group	65	37	MicroMotion	73	75	Solution Systems
17	C Ware		38	Microprocessors Unlimited	113	62	The Source View Corporation 113
*	DDJ Change of Address		39	Micron Technology, Inc	65	61	Southern Computer Corporation 97
70	DDJ Advertise		40	Morgan Computing Company, Inc	. 1	63	Supersoft
71	DDJ Back Issues		41	Opt-Tech Data Processing	73	64	Telecon Systems 51
72	DDJ Bound Volume		42	Overbeek Enterprises	41	65	Western Wares113
73	DDJ Subscription		43	Phlexible Data Systems	115	66	Mark Williams & Company 2
18	Data Access Corporation		44	Phoenix Software		67	Wordtech Systems, Inc
19	Datalight		45	Proa Corporation		68	Workman & Associates128
20	D & W Digital		46	ProCode International			
21	Ecosoft, Inc.		47	The Programmer's Shop			
21	Leosoft, Ille						



LATTICE®

C Compilers

'My personal preferences are Lattice C in the top category for its quick compile and execution times, small incremental code, best documentation and consistent reliability;...

BYTE AUG. 1983

R. Phraner

"... programs are compiled faster by the Lattice C compiler, and it produces programs that run faster than any other C compiler available for PC-DOS."

PC MAGAZINE JULY 1983

H. Hinsch

"... Microsoft chose Lattice C both because of the quality of code generated and because Lattice C was designed to work with Microsoft's LINK program."

PC MAGAZINE OCT. 1983

D. Clapp

"Lattice is both the most comprehensive and the best documented of the compilers. In general it performed best in the benchmark tests.

PERSONAL COMPUTER AGE NOV 1983 F. Wilson

"This C compiler produces good tight-running programs and provides a sound practical alternative to Pascal."

SOFTALK AUG 1983

P. Norton

"... the Lattice compiler is a sophisticated, high-performance package that appears to be well-suited for development of major application programs."

BYTE AUG 1983 Houston, Brodrick, Kent

To order, or for further information on the LATTICE family of compilers, call or write:



(312) 858-7950

LATTICE, INC. P.O. Box 3072 Glen Ellyn, IL 60138





Circle no. 31 on reader service card.



WRITE

The Writer's Really Incredible Text Editor lives up to its name! It's designed for creative and report writing and carefully protects your text. Includes many features missing from WordStar, such as sorted directory listings, fast scrolling, and trial printing to the screen. All editing commands are single-letter and easily Detailed manual included. changed. WRITE is \$239.00.

WORKMAN & ASSOCIATES

112 Marion Avenue Pasadena, CA 91106 (818) 796-4401

All US orders are postpaid. We ship from stock on many formats, including: 8", Apple, Osborne, KayPro, Otrona, Epson, Morrow, Lobo, Zenith, Xerox. Please request our new catalog. We welcome COD orders.

Circle no. 68 on reader service card.

We thought about calling it MacSimplex . . . after all it makes your IBM®PC behave like a Macintosh™ and much more . . .

and with over two years in the making, the Simplex Database Management System has features like 32-megabyte virtual memory and the most powerful networked/relational database in the microcomputer industry. Simplex was designed around how you think and the Macintosh way, so that you can use your favorite mouse to handle those mundane tasks like menu selection and data manipulation. And, if you don't have a mouse, you can use our keyboard mouse simulator, MouSim $^{\text{TM}}$.

Pop-up and pull-down menus, dialog and alert boxes are not just added features, they are the heart of the Simplex way. In addition, Simplex gives you both a software and a hardware floating point capability, each with 19-digit accuracy. It permits login, password, privilege, and can be used on a local area network. Simplex has full communications and a remote or local printer spooler. Above all, Simplex is modular and grows with you! Simplex also has a full-featured, English-like language which is simple to use.



You can't buy Simplex[™], but it is now available as an integral part of it's my **Business**[™] and will be used by it's my **Word**[™], it's my **Graphics**[™], . . .

Businessmen! *it's my* **Business** will revolutionize the way that you handle your business. It saves time, money, and standardizes your system for all who use it. *it's my* **Business** comes with applications like accounting, interoffice or intraoffice mail, editing, invoicing, inventory managment, mail list, calendar, scheduler, forms and more. You can modify each of these to create applications specifically designed for you... maybe we should have called it "it's your Business".

Professionals! it's my **Business** has over 200 pages of examples and demonstrations to show you how to solve your everyday professional problems. And if these examples aren't enough, we give you a complimentary one-year subscription to Questalk $^{\text{\tiny TM}}$, our hands-on Simplex applications magazine.

System integrators and consultants, beware! If you are not using *it's my Business* with Simplex to solve your problems, don't be surprised when more novice programmers solve that complex math, industrial engineering, or business problem faster. We think that you can cut your concept-to-development time by an order of magnitude!

it's my **Business** (includes it's my **Editor**) - \$695.00 it's my **Business** Demo Disk - \$20.00 it's my **Editor** - \$100.00.

Quest Research software is available through your local computer store or through mail order from Quest Software Corporation at (205) 539-8086. 303 Williams Avenue, Huntsville, AL 35801.

Value added resellers and dealers please contact Quest Research, Incorporated at (800) 558-8088. 303 Williams Avenue, Huntsville, AL, 35801.



IBM is a registered trademark of International Business Machines. Macintosh is a trademark of Apple Corporation. it's my Business, it's my Word, it's my Graphics, it's my Home, it's my Voice, it's my Statistics, Simplex, MouSim, Questalk, and the Quest logo are trademarks of Quest Research, Incorporated.

This is THE PASCAL COMPILER You've Been Hearing About



"It's almost certainly better than IBM's Pascal for the PC.. Recommended." **Jerry Pournelle** Byte, May 1984

\$49.95

"If you don't have CP/M [for your Apple], Turbo Pascal is reason enough to buy it." Cary Hara

Softalk Apple, May 1984

"If you have the slightest interest in Pascal . . . buy it." Bruce Webster, Softalk IBM, March, 1984

And Now It's Even Better Than You've Heard!

- Windowing (IBM PC, XT, ir. or true compatibles)
- Color, Sound and Graphics Support (IBM PC, XT, jr. or true compatibles)
- Optional 8087 Support (available at an additional charge)
- Automatic Overlays
- A Full-Screen Editor that's even better than ever
- Full Heap Management—via dispose procedure
- Full Support of Operating System Facilities
- No license fees. You can sell the programs you write with Turbo Pascal without extra cost.

Yes. We still include Microcalc . . . the sample spreadsheet written with Turbo Pascal. You can study the source code to learn how a spreadsheet is written . . . it's right on the disk.* And, if you're running Turbo Pascal with the 8087 option, you'll never have seen a spreadsheet calculate this fast before!

*Except Commodore 64 CP/M.

Order Your Copy of TURBO PASCAL® VERSION 2.0 Today

For VISA and MasterCard orders call toll free: In California: 1-800-227-2400 x968 1-800-772-2666 x968

(lines open 24 hrs, 7 days a week)

Dealer & Distributor Inquiries Welcome 408-438-8400

Choose One (please add \$5.00 for shipping and handling for U.S. orders. Shipped UPS)

Turbo Pascal 2.0 \$49.95 + \$5.00

Turbo Pascal with 8087 support \$89.95 + \$5.00

Update (1.0 to 2.0) Must be accompanied by the original master \$29.95 + \$5.00

Update (1.0 to 8087) Must be accompanied by the original master \$69.95 + \$5.00

Money Order Check _ VISA -Master Card Card #: Exp. date:



Borland International 4113 Scotts Valley Drive Scotts Valley, California 95066 TELEX: 172373

Circle no. 8 on reader service card.

My system is: 8 bit ___ 16 bit _ Operating System: CP/M 80 _ CP/M 86 __ MS DOS __ PC DOS _ Computer: Disk Format:

Please be sure model number & format are correct. Name: Address:

City/State/Zip: _ Telephone:

California residents add 6% sales tax. Outside U.S.A. add \$15.00 (If outside of U.S.A. payment must be by bank draft payable in the U.S. and in U.S. dollars.) Sorry, no C.O.D. or Purchase Orders. D11